

P'X5 Sales Solution

Authoring Workbench User's Guide

Version 9.0

Copyright

© Copyright 2015 Perspectix AG. All Rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the written permission of Perspectix AG. The information contained herein may be changed without prior notice. Perspectix AG shall not be liable for errors or omissions with respect to this publication.

P'X™, P'X5™ and the corresponding logos are registered trademarks of Perspectix. Other company, product, and service names may be registered trademarks of others.

Perspectix AG

Hardturmstrasse 253
CH-8005 Zurich
Switzerland

Phone: +41 44 445 95 95

Fax: +41 44 445 95 96

Website: www.perspectix.com

Email: info@perspectix.com

CONTENT

Overview	1
Installation and Upgrade	2
Install the Authoring Workbench	2
Install Unsupported PXML Reference Target	7
Uninstall the Authoring Workbench	10
Upgrade the Authoring Workbench	11
Teamwork with CVS	13
User Interface and Interaction	20
Views	21
Project Explorer	23
Part Navigator	24
Outline	24
Expression View	26
Problems View	27
Search View	28
VCML Hierarchy View	30
Polyedit	31
Call Hierarchy View	31
Editors	32
Customization Dashboard	34
Nodedataviewer Editor	35
Nodesystem Editor	36
Rule Editor	39
Translation Editor	39
Pages	40
Node Inspector	44
Inspector	47
Preferences	49
Tutorials	53
Basic Tutorial (Practice)	54
Lesson 1 - Create Components	55
Lesson 2 - Create Geometries and Assign them to Components	61
Lesson 3 - Create Docks	69
Lesson 4 - Assign Docks to Components	79
Lesson 5 - Create Icons and Assign them to Components	86
Lesson 6 - Define Materials and Assign them to Geometries	100
Lesson 7 - Create Properties and Assign them to Geometries	105
Lesson 8 - Create Assemblies and Rules	118
Lesson 9 - Create Article List	132
Lesson 10 - Insert Properties into Article List	144
Cheat Sheets	152

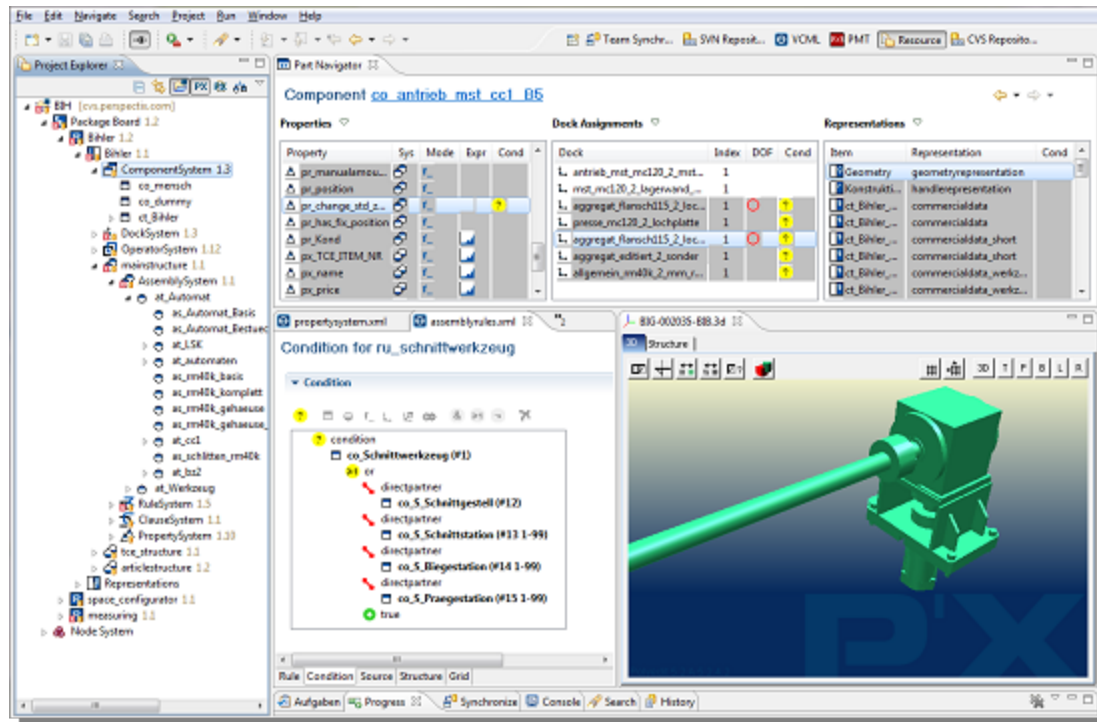
Geometry Requirements	153
Common Preferences	155
Export VCML Documentation	158
Manage Include Files for Parts	161
Merge LODs with Polyedit	163
Polyedit License Handling	169
Select Built-In Target Platform	170
Refactor Expressions	171
Replace PXML Schemas	174
Translate User Interface Language	175
Tips and Tricks	181

Overview

The Authoring Workbench is the authoring tool for writing VCML.

VCML (Visual Configuration Modeling Language) is an XML (eXtensible Markup Language) dialect, which is used to drive and configure the Project Organizer and the Configurator, but also serves as an information repository for product metadata.

In the Authoring Workbench you can create product information relating to catalogs, calculation formulas, and rules for the assembly logic and the parts list generation.



Installation and Upgrade

Note: The version 8.3 of the Authoring Workbench requires a new installation. An update from a older version is not possible. Please make sure, that all previous versions are uninstalled, before you install version 8.3.

To install AWB Version 8.3:

- Rename the installation folder `PX5 Authoring Workbench` in the directory `C:\Program Files (x86)\Perspectix`, so you can still start this version of the AWB in case something went wrong with the new installation.
- Install Version 8.3 with administrative privileges.

Install the Authoring Workbench

Note: To install the Authoring Workbench, you need Java Runtime Environment (JRE) 1.5 or higher (www.java.com/getjava/).

To install the Authoring Workbench

1. Download the `PX5_AWB_Setup.exe` to your hard drive.
2. Run the downloaded executable file to start the installation process.
3. Depending on your setup, additional steps may be required to complete the setup. Ask your Perspectix coach to help with the following steps if needed:
 - Install the license key obtained from Perspectix.
 - Check out your P'X5 project from CVS.

Note: The newest target platform versions are included in the Authoring Workbench. You can also install the old and unsupported target platform versions. See page 7.

Installation without Administrative Privileges

Perspectix recommends that the Authoring Workbench is installed for all users on the system, which requires administrative privileges.

If the installer is executed by a user without administrative privileges, the installer will propose to install the Authoring Workbench into the user's area. This entails the following drawbacks:

- The Authoring Workbench is only available to the current user.
- The HASP dongle driver will not be installed, since it requires administrative privileges. The driver must be installed separately or already be present on the target system.

Installing the Authoring Workbench from a Zip File

The zip build contains the same content as the executable installer package, but it does not create a start menu launcher nor an entry in the Installed Software list of the operating system.

Use the zip build to easily maintain different installations of the Authoring Workbench for testing purposes without modifying the Windows registry. You can install the zip build without administrator privileges.

To install the Authoring Workbench from a zip file

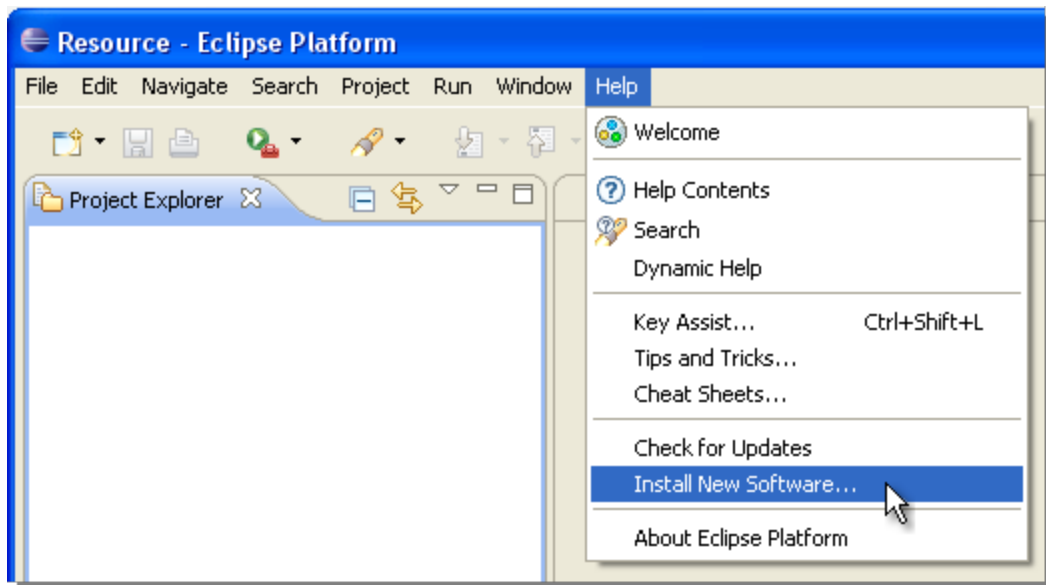
1. Download the `awb.zip` file to a location on your hard drive.
2. Create a folder to hold the installation in any location where you have write permissions, e.g. `My Documents\awb`.
3. Extract the downloaded zip file to the folder you created.
4. Run the `awb.exe` file in the target folder.

Installing the Authoring Workbench into an Existing Eclipse Product

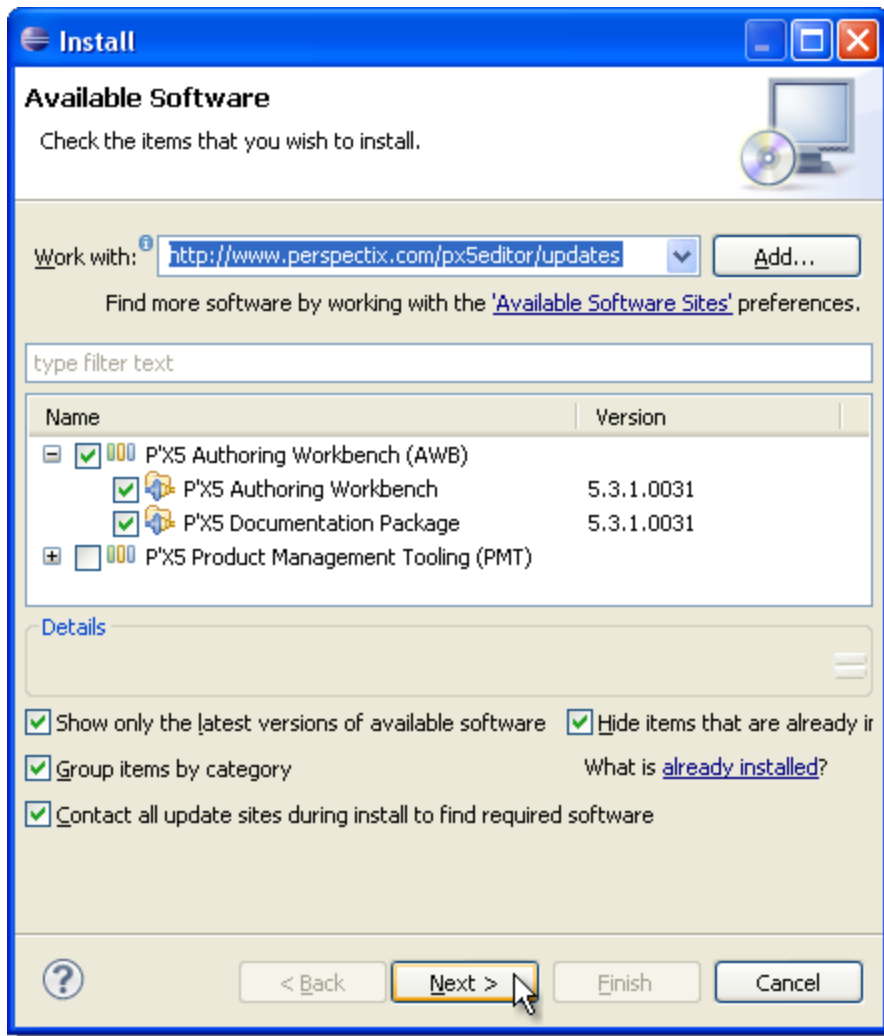
If you already have a product based on Eclipse 3.5.1 or greater installed, e.g. for Java or web development, you can install the Authoring Workbench as part of that product.

To install the Authoring Workbench into an existing Eclipse product

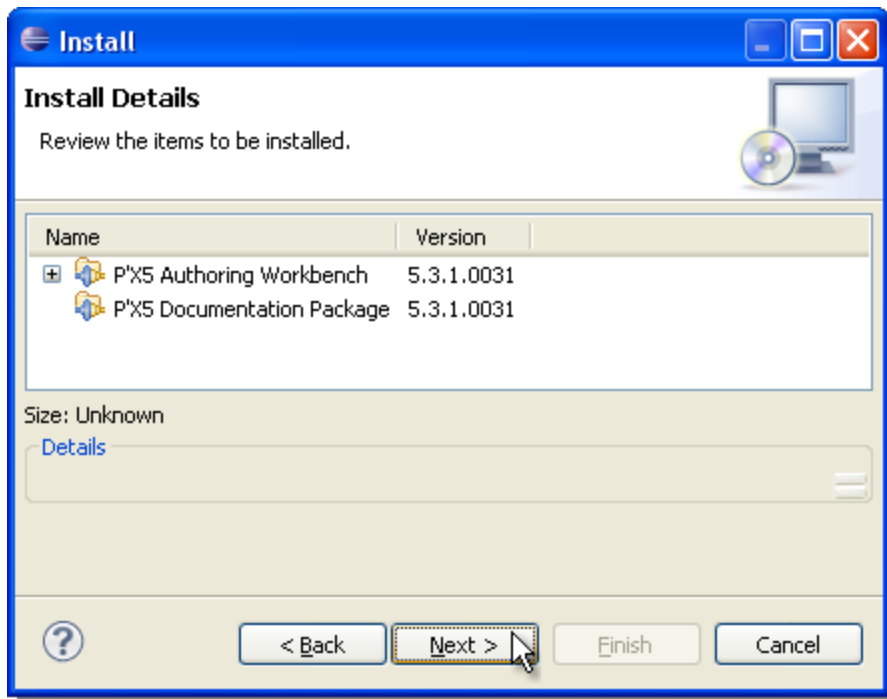
1. Start the Authoring Workbench.
2. Choose **Help > Install New Software** from the main menu.



3. Type the following URL <http://www.perspectix.com/px5editor/updates> in the **Work with** field in the **Install** window.
4. Select all items under **P'X5 Authoring Workbench (AWB)** and click **Next**.



5. Review the items to be installed and click **Next**.



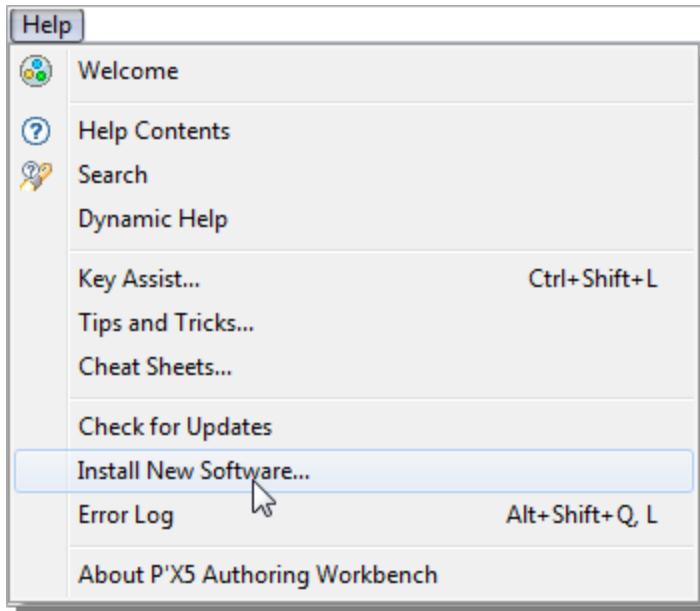
6. If necessary, review the licenses and click **Finish**.
7. Wait for the installation process to download all required plugins.
8. Click **Yes** when asked to restart the Authoring Workbench.
9. When the Authoring Workbench has restarted, choose **Window > Open Perspective > Others > VCML** to open the default P'X5 perspective.

Install Unsupported PXML Reference Target

The target platform versions 6.0, 6.1, 6.2, and 7.0 are included in the Authoring Workbench. You can also install the old and unsupported target platform versions 5.2 and 5.3.

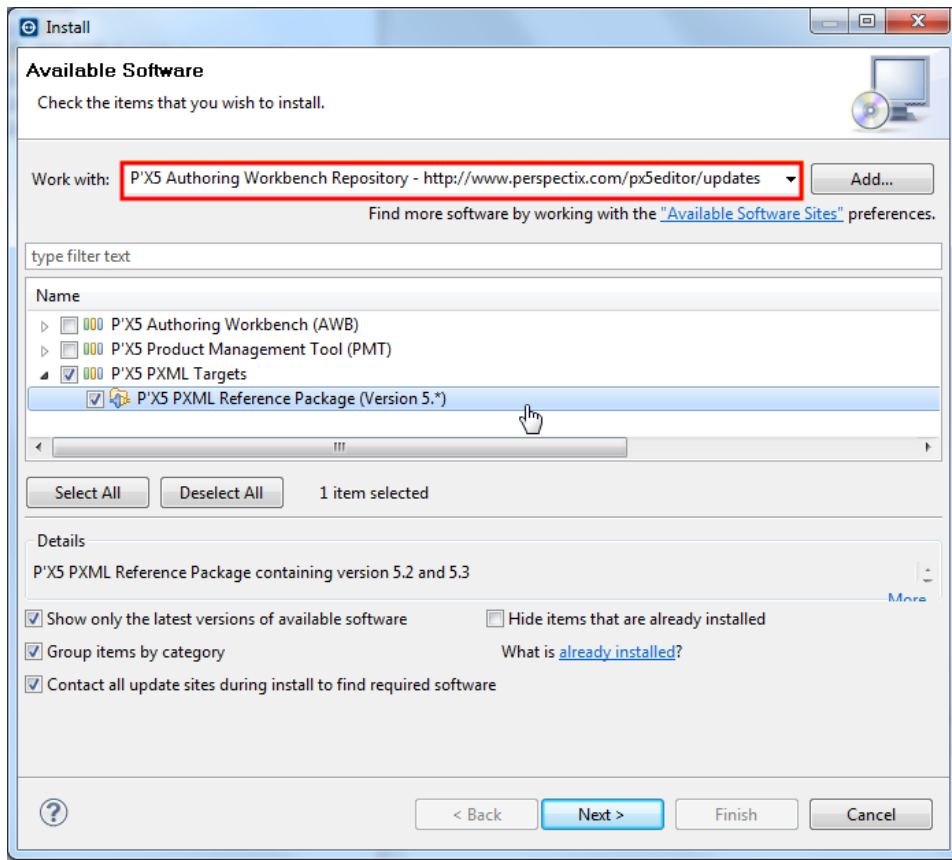
To install the target platforms

1. Start the Authoring Workbench.
2. Choose **Help > Install New Software** from the main menu.

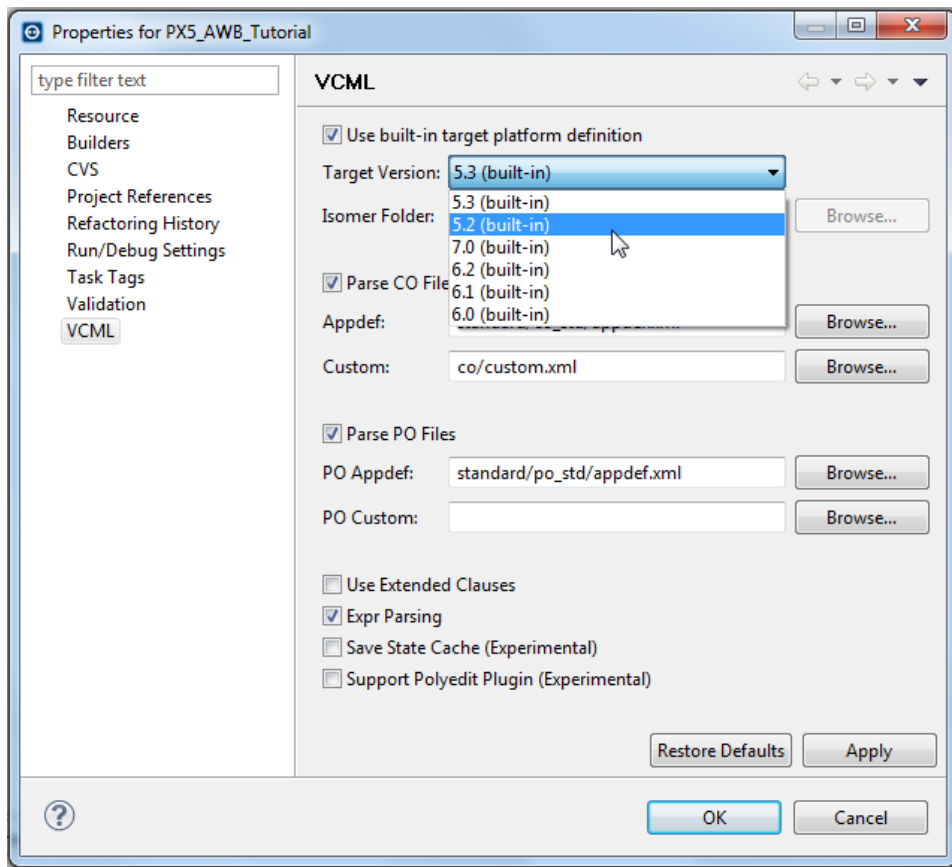


3. Select the Authoring Workbench update site <http://www.perspectix.com/px5editor/updates> in the **Work with** field in the **Install** window.
4. Select the target platform you want to install and click **Next**.

Note: The target platform plugins are packaged for each major version (i.e. 5.0, 6.0, 7.0, etc.). As the target mechanism was introduced in version 6.0, only one unsupported package (5.*) is currently available.



5. Review the items to be installed and click **Next**.
6. Click **Finish** to close the wizard.
7. Wait for the installation process to download the target platform.
8. Click **Yes** when asked to restart the Authoring Workbench.
9. When the Authoring Workbench has restarted, the target platforms are automatically available in the **Properties** window of the project.



Uninstall the Authoring Workbench

To uninstall the Authoring Workbench

1. Open the Windows **Control Panel**.
2. Open the **Add or Remove Programs** page and select to uninstall the Authoring Workbench.

Note: You may require administrative privileges if the Authoring Workbench was installed system-wide.

Upgrade the Authoring Workbench

If you already have a recent installation of the Authoring Workbench, you can upgrade to the newest version without running the installer again.

Prerequisites

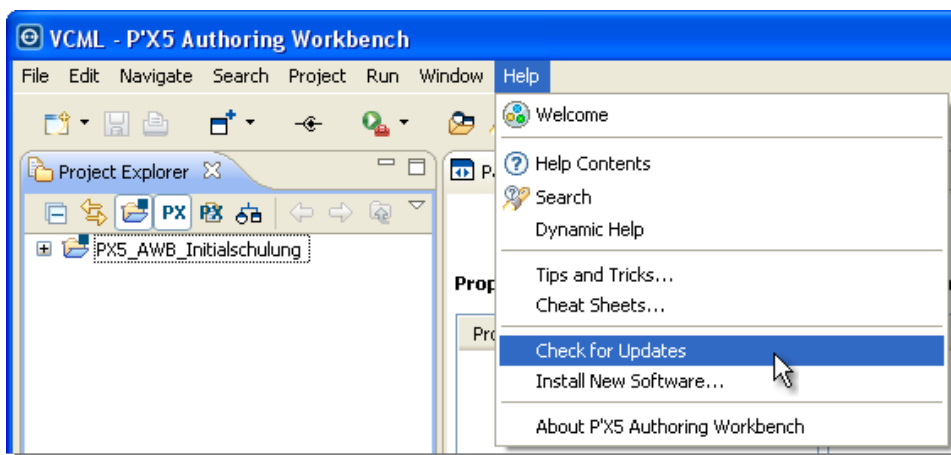
- Authoring Workbench version 5.3 or newer is already installed. To see the installed version, choose **Help > About**.

Note: If your Authoring Workbench is older than version 5.3, you must first uninstall the Authoring Workbench and then reinstall a fresh installation. See pages 10 and 2.

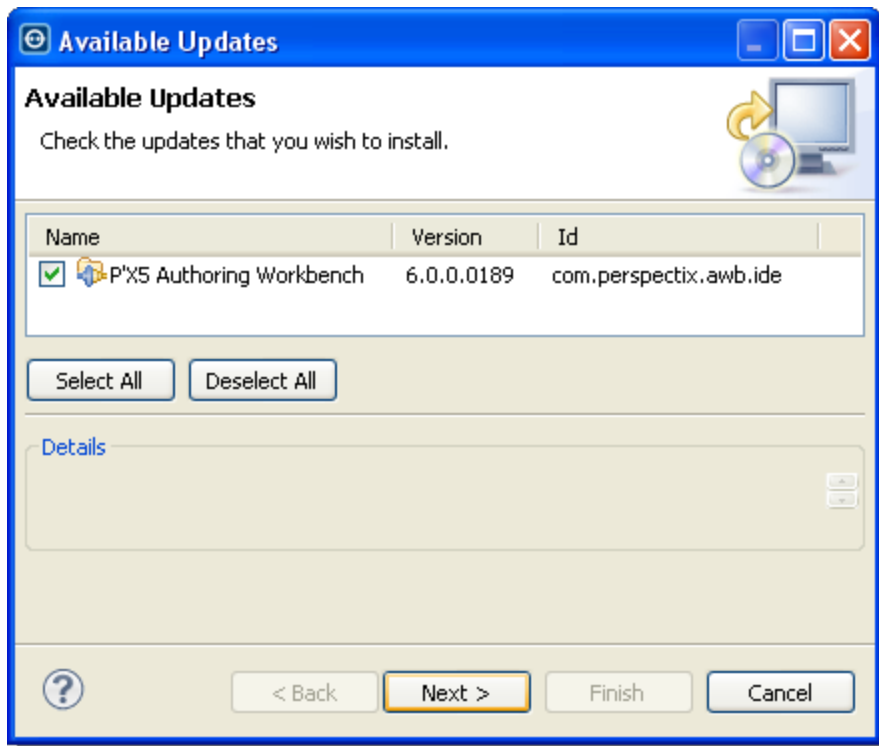
- Administrative privileges are available.
- A working internet connection is available.

To upgrade the Authoring Workbench

1. Start the Authoring Workbench with administrative privileges (on Windows Vista choose **Run as administrator** from the link's context menu).
2. Choose **Help > Check for Updates** from the main menu.



3. In the **Available Updates** window select the updates that you want to install and follow the wizard to complete the update.



4. Wait until the updates have been downloaded and installed.
5. Click **Yes** when asked to restart the Authoring Workbench.

Teamwork with CVS

Creating a new project

- Under File, select "New > Projects from CVS"
- Fill in the following mask.
 - Host: Address of the Knowledge Base Server

Note: Both the update server and the CVS server (Knowledge Base Server) can be reached at the same address.

- Repository Path: /srv/px5-cvsroot
- User: cvs user
- Password: cvs user password
- Connection Type: extssh
- Use default port

Checkout from CVS

Enter Repository Location Information

Define the location and protocol required to connect with an existing CVS repository.

Location

Host:

Repository path:

Authentication

User:

Password:

Connection

Connection type:

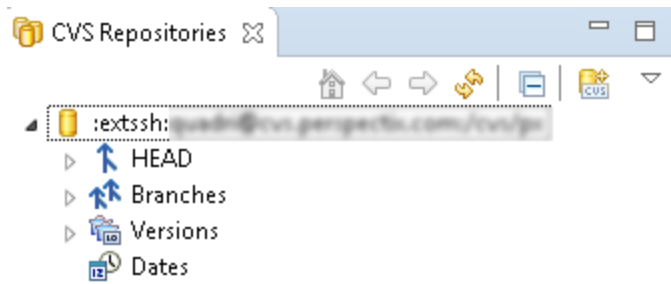
☒ Use default port

☐ Use port:

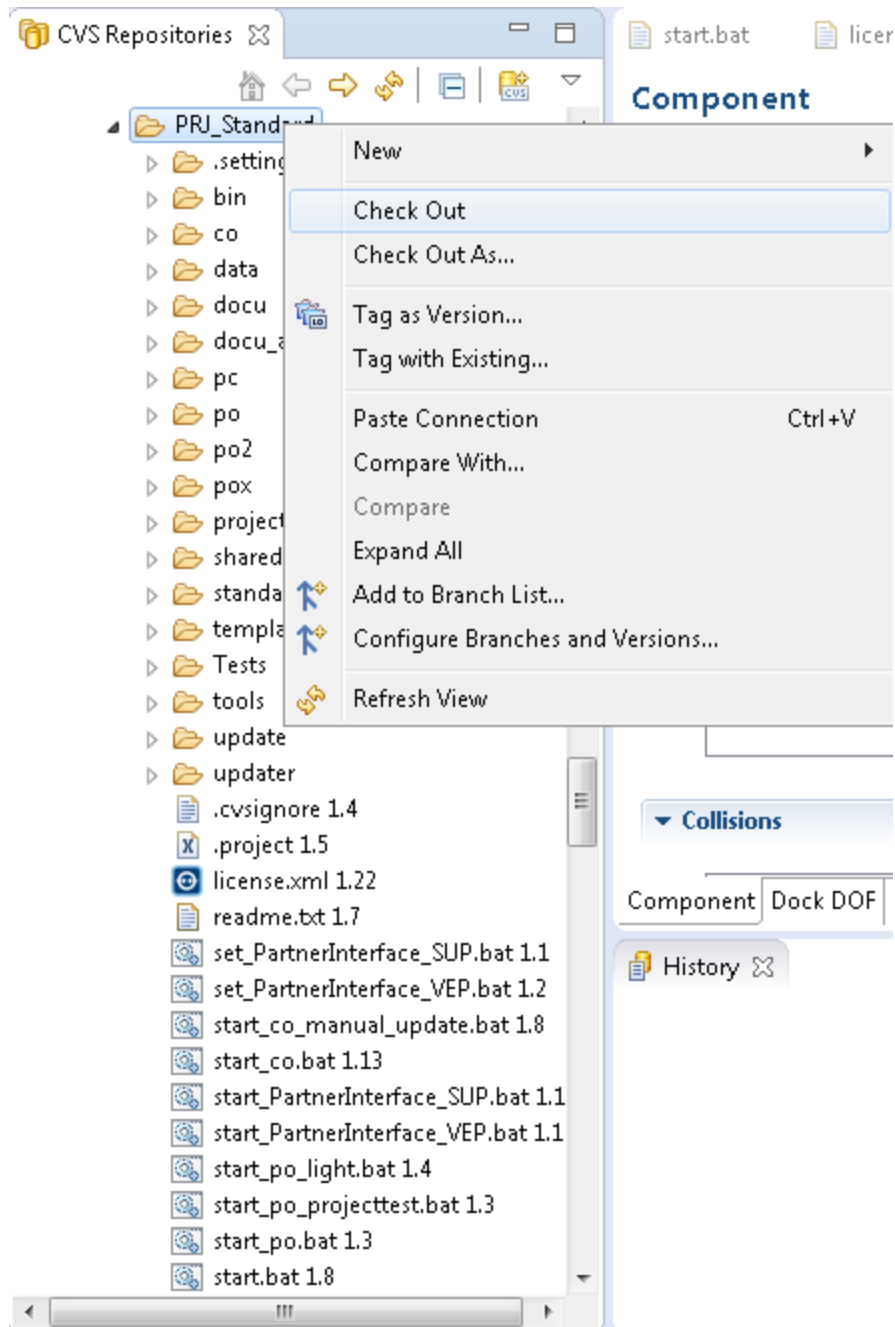
☐ Save password (could trigger secure storage login)

To manage your password, please see ['Secure Storage'](#)

- Click "Finish" to establish the connection.
- Go to the perspective "CVS Repository Exploring". Now you have access to the project data. You can find the current database under HEAD.



- Open the area under HEAD and select the project. Open the context menu (use a right mouse click) and check the project out.

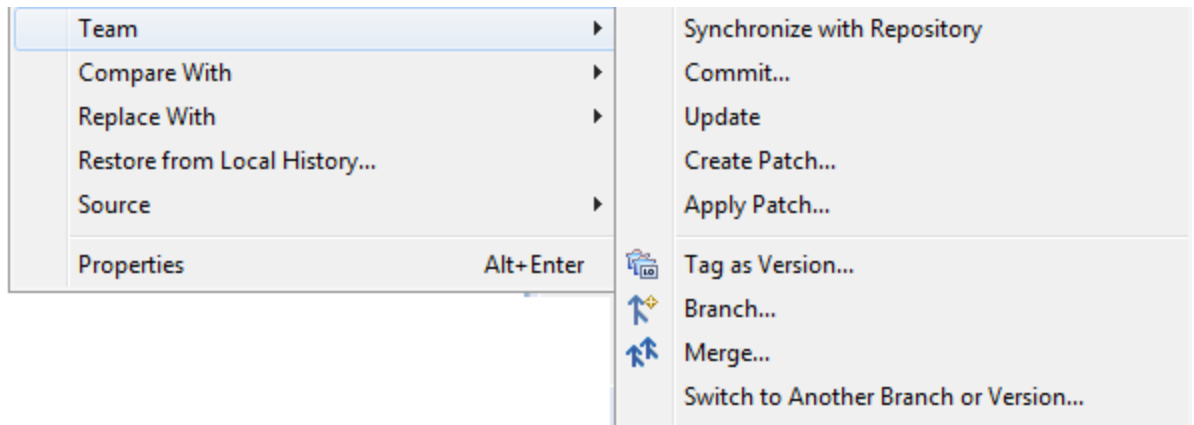


- A local copy of the project is created in your workspace.

Now you can work on the project.

Synchronizing data with the CVS server

Changes you make to the project only exist locally at first. You must check the changes in so that they become available on the CVS server. It is also possible that other authors who have been working on the project as well may also check in their changes.

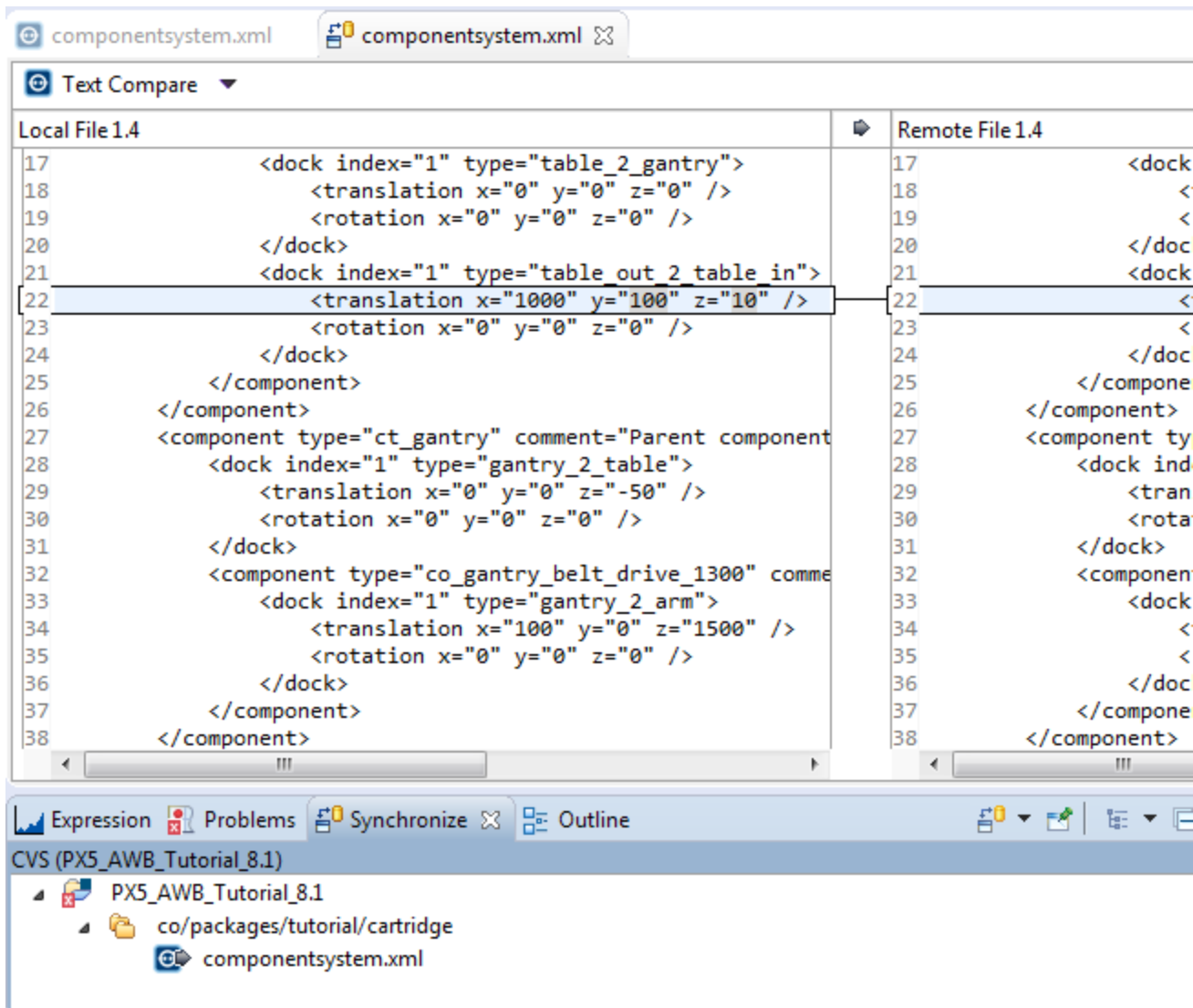


To determine the current status of the project, synchronize it with the version on the server.

- Select the project in Project Explorer and choose the option "Team - Synchronize with Repository" in the context menu.

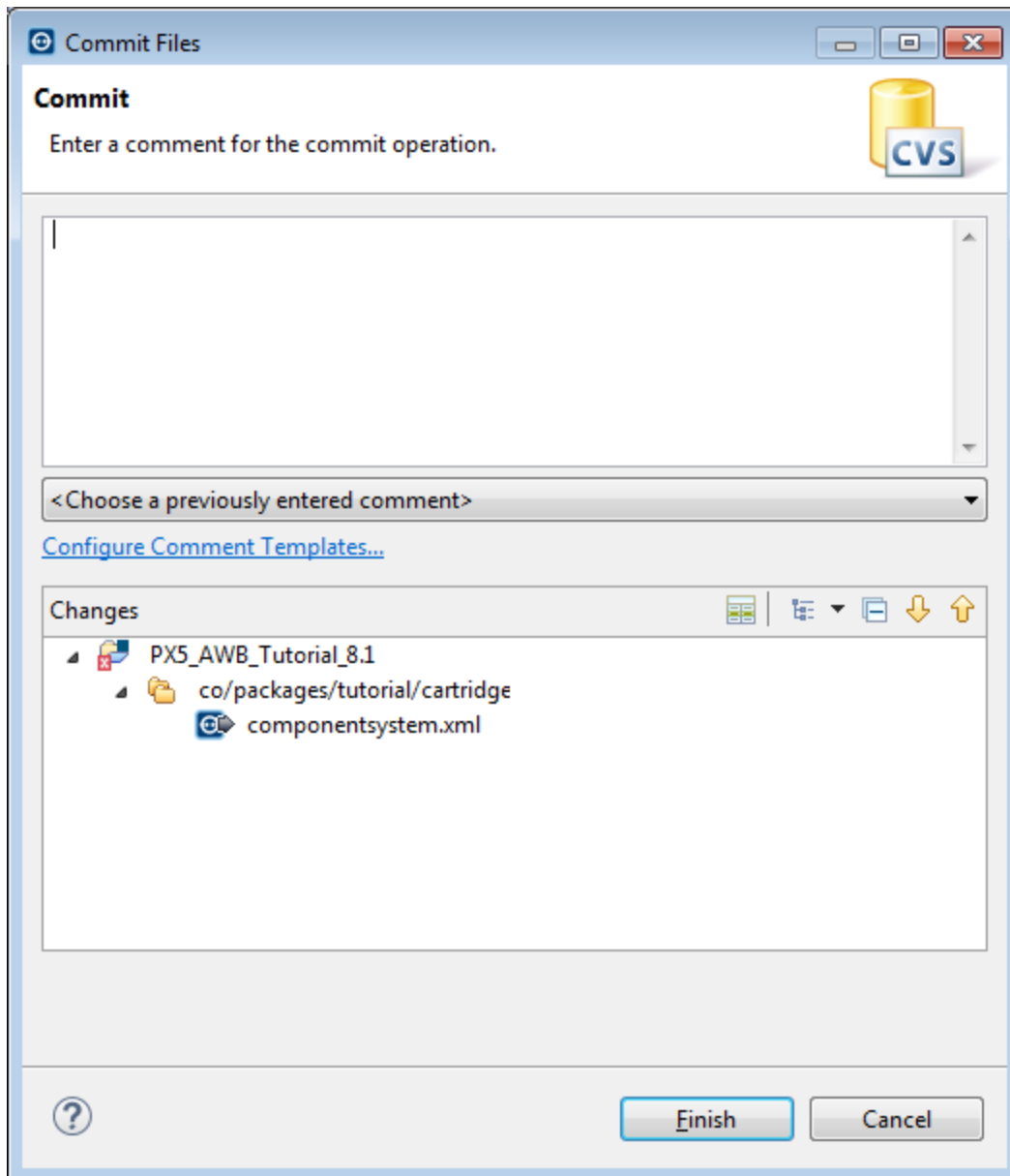
The local version of the project is then compared with the version on the server. You are shown the changes to the project in the Synchronize view. A distinction is made here between incoming and outgoing changes.

- Double click on a file in the "Synchronize" view in order to see the differences between the local version and the server version. This opens a comparison view. Changes within a file can be seen here.



- To apply the changes from the server locally, select the project in Project Explorer and choose the option "Update" in the context menu. All of the changes from the server are saved locally.
- To apply your own changes to the server, these must be committed. Select the project in Project Explorer and choose the option "Commit" in the context menu.

Enter a comment in the following window. Comments should be as meaningful as possible and make sense for co-authors. If several authors are working on a single project, it has proven helpful to place your own initials in front of each comment. This makes it easier to recognize who has made the changes.



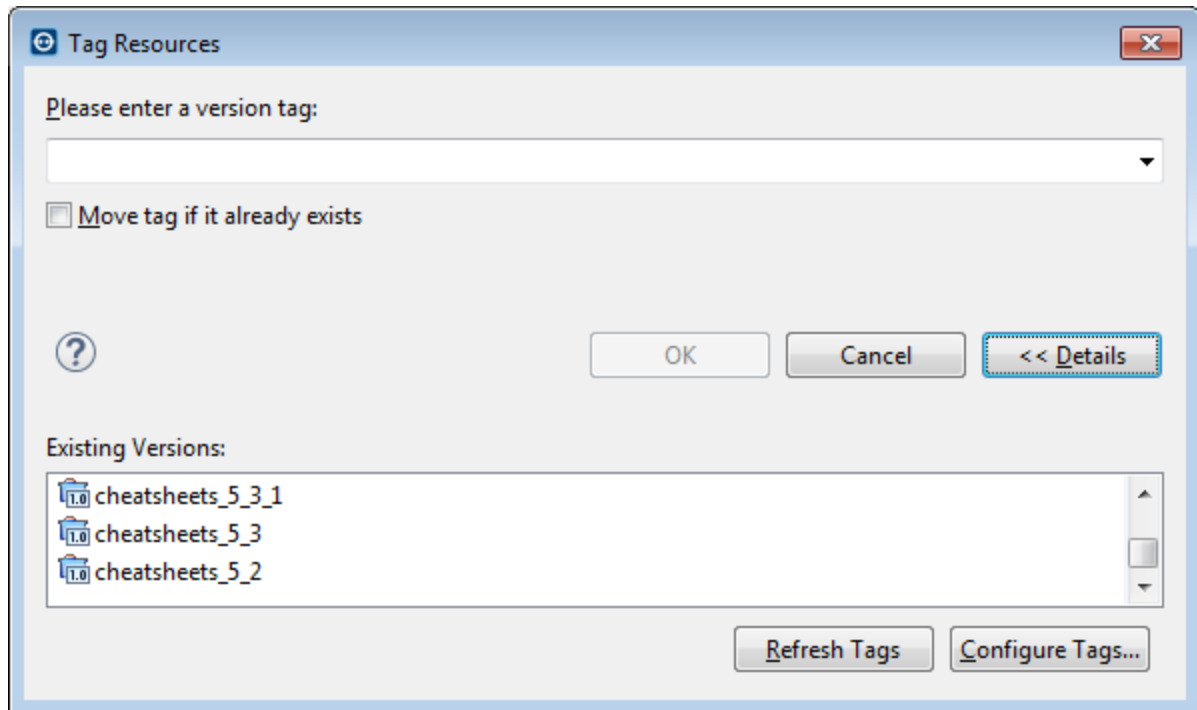
Note: It is not necessary to always update or commit the entire project. Instead of selecting the entire project, only select one folder or a file.

Working with versions

If a status has been reached in the project that you would like to distribute for testing or for productivity purposes, proceed as follows:

- Ensure that all data are up to date and your local changes have been checked in.
- Ensure that the project runs correctly and no errors occur.
- Create a version using the current status. Select the project in Project Explorer and choose the option "Team - Tag as Version" in the context menu.

- Give the version a meaningful name.
- You can have already existing versions shown so that you can comply with naming conventions.



Note: Creating a version can also make sense to secure the status of a project. This is useful, for example, before major changes are made.

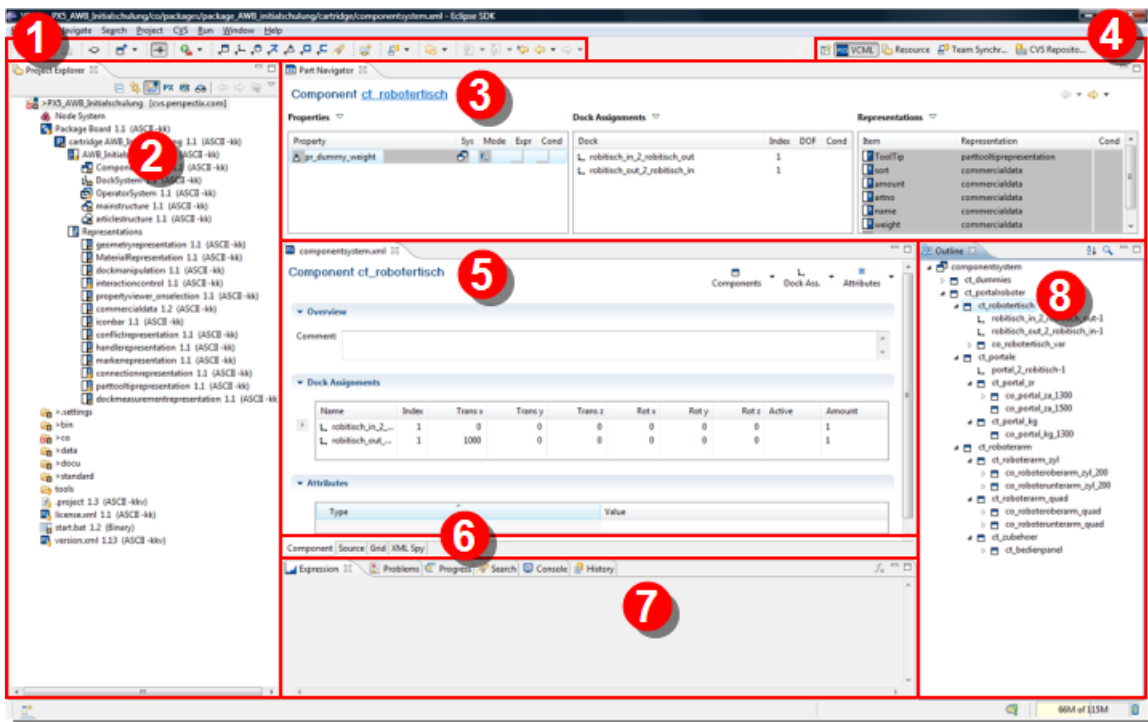
Information about creation of the Software Installer and Software Updates can be found under software distribution.

User Interface and Interaction

The Authoring Workbench is based on Eclipse. If you have experience with Eclipse, you will immediately find yourself familiar with the VCML Perspective and the Workbench layout.

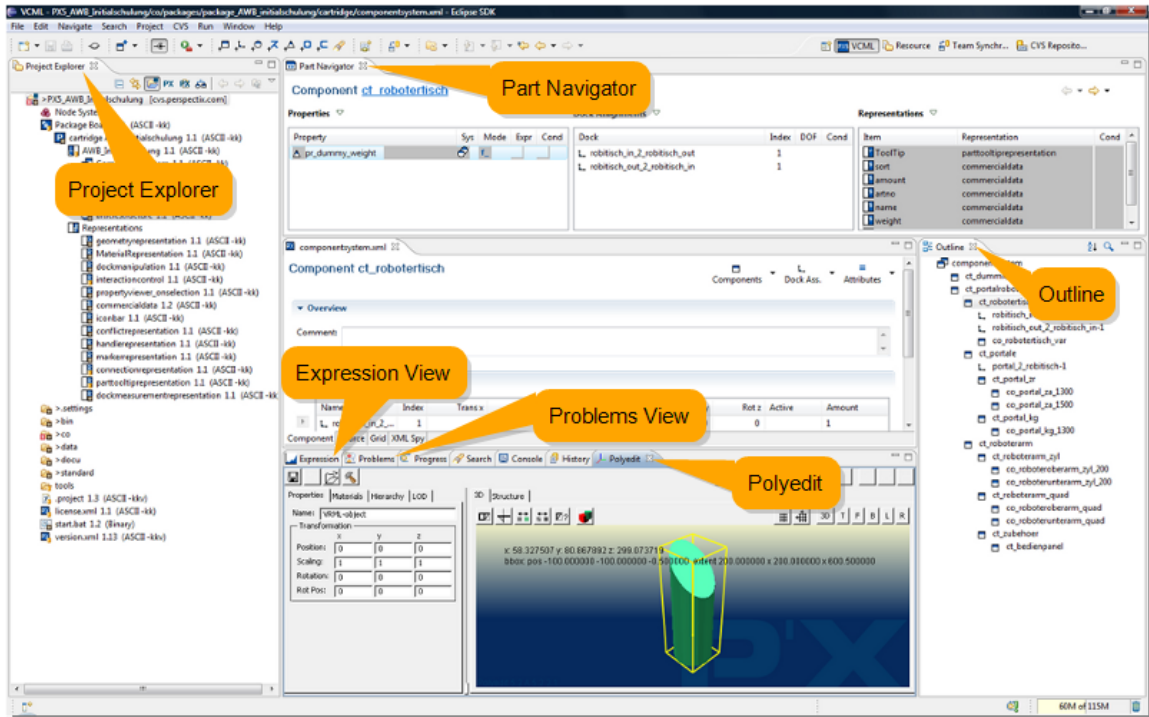
Tip: For a detailed description of the Eclipse user interface and interaction, see the Workbench User Guide in the Eclipse Online Help.

1. In the **Toolbar** you create and open VCML elements.
2. The **Project Explorer** (View) shows the structure of the project. The classification tree shows all created parts.
3. The **Part Navigator** (View) shows the referenced elements of the active part.
4. In the **Perspectives** you open the VCML Perspective.
5. The **Editors** contain clearly laid out fields and input assistance.
6. On the **Pages** you edit the XML code (**Source Page**), data in tables (**Grid Page**), DOF definitions (**Dock DOF Page**), etc.
7. In the **Views** you use VCML expressions (**Expression View**), view warnings and syntax errors (**Problems View**), edit and illustrate geometries of components (**Polyedit**), etc.
8. The **Outline** (View) shows a structural overview of the open file.



Views

You can quickly navigate between views and display a VCML element in a different context.



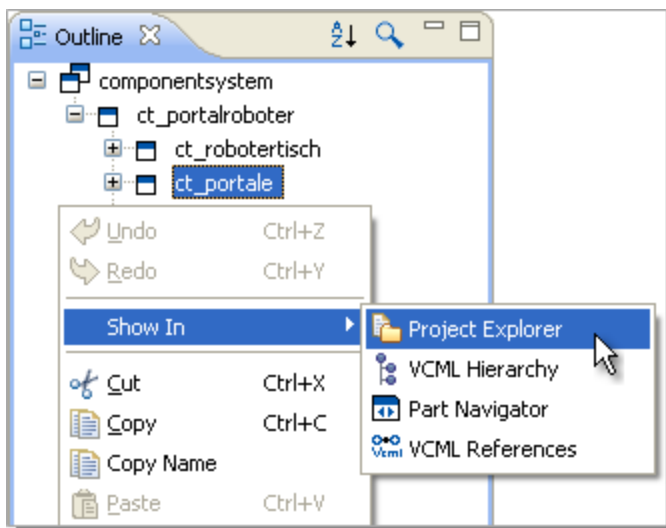
To open a View

- Choose **Window > Show View** from the main menu and then choose the desired view.

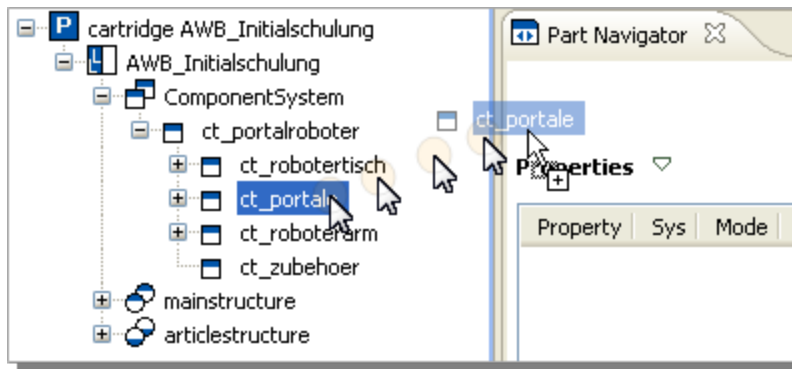
To display an element in another view

Do one of the following:

- Right-click an element and choose **Show In** and the view from the context menu.



- Choose **Navigate > Show In** and the view in the main menu.
- Drag an element from a view and drop it into another view.



Project Explorer


The Project Explorer shows the structure of a project.

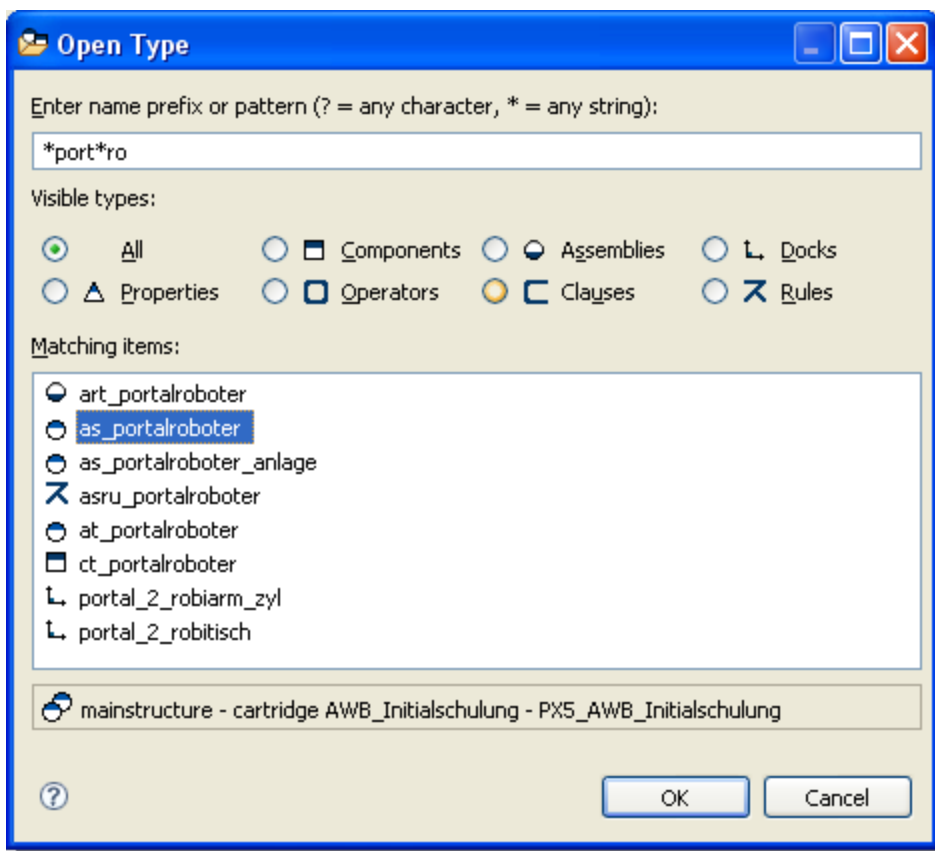
The Part Tree in the Project Explorer shows all parts (components, assemblies of all structure systems) in a product knowledge base. The components and assemblies are displayed as classification hierarchies in tree structures. Double-clicking a part in the part tree updates the Part Navigator to the right of it.

Open VCML Types

You can directly open a VCML type of which you remember the name, instead of tediously navigating to the desired type in the Project Explorer.

To open a VCML type

1. Click the Open a VCML Type icon  in the main toolbar or choose **Navigate > Open VCML Type** from the main menu.
2. Enter the name of the type in the **Open Type** window and click **OK**.




Tip: Press CTRL+T to open the **Open Type** window.

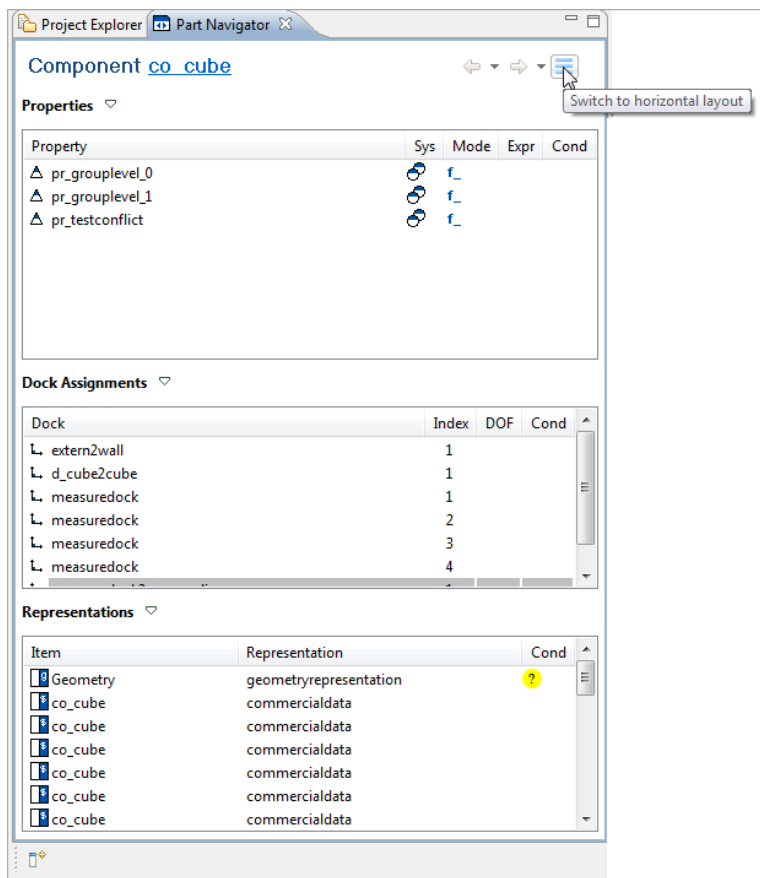
Part Navigator

The Part Navigator displays an overview of all relevant information about a product element. Properties, docks, and representations are listed under components, and properties, assembly rules, and representations are listed under assemblies.

Using drag & drop, you can add new properties to a product element and maintain product data using context-sensitive menus. Double-clicking a list entry in the Part Navigator opens the appropriate editor. You can allocate representations, such as 3D geometries, catalog entries or interaction handles to parts or generate them anew.

To switch between the horizontal and vertical layout

- Click the layout icon  in the upper right hand corner of the Part Navigator.



Outline

The Outline shows a structural overview of the open file.

Outline of the Named Operations Editor

The Outline of the Named Operations Editor now shows decorations for the different states. This helps to identify operations quicker via the VCML Content Outline.

State	Description	Decoration
Overwrites	The <i>Named Operation</i> overwrites an existing one.	Green arrow at the bottom right
Final	The <i>Named Operation</i> is marked as final.	"F" on the top right corner
Deprecated	The <i>Named Operation</i> is marked as deprecated.	Diagonal line in the background

State	Description	Decoration
Error	VCML Error on <i>Named Operation</i>	Error icon at the bottom left
Combined	Combination of two or more states	Combined decorations

namedoperations_overwrites.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<namedoperations xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" x
  <namedoperation name="overwrites">
    <!-- Overwriting Named Operation -->
  </namedoperation>

  <namedoperation name="final" final="true">
    <!-- Final Named Operation -->
  </namedoperation>

  <namedoperation name="deprecated" deprecated="true">
    <!-- Final Named Operation -->
  </namedoperation>

  <namedoperation name="error">
    <!-- Named Operation with error -->
  </namedoperation>

  <namedoperation name="combined" deprecated="true" final="true">
    <!-- Named Operation with error -->
  </namedoperation>

</namedoperations>

```

Outline

- namedoperat
- overwrite
- final()
- deprecate
- error()
- combined

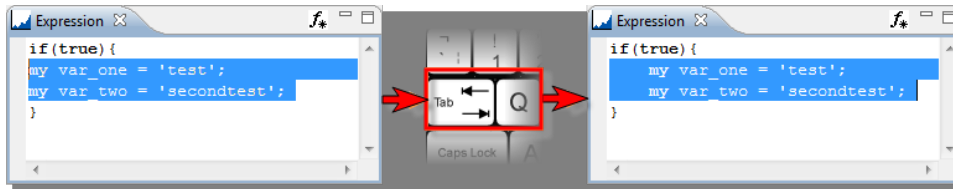
Expression View

The Expression View offers input assistance and VCML expressions.

The Expression View supports the script declaration by means of syntax highlighting, code completion, on-the-fly error display and integrated syntax help. The functions, which are selected in the pop-up window of the code completion or in the function browser can be inserted into the script with a single click. In addition to calculations, you can also specify functions and sequencing logic in the Expression View in order to define automatisms for the application logic and workflows for the system integration.

To indent multiple lines of code

- Select the code and press TAB. To undo, press SHIFT+TAB.



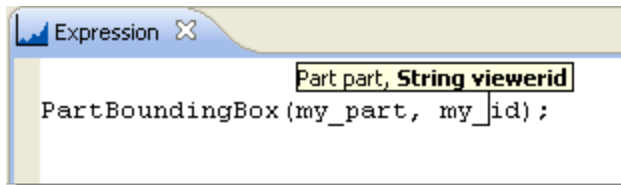
Version: Requires P'X5 version 6.2 or higher.

Parameter Information of Expression Functions

After applying a proposal for an expression function, context information about its parameters is displayed.

To manually display the parameter information

- Press CTRL+SHIFT+SPACE within the argument list of a function.



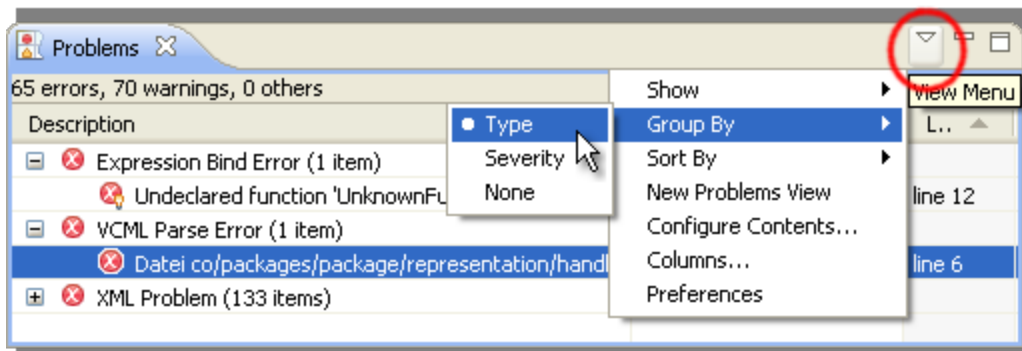
Problems View

The Problems View shows problems and errors in the current project or the entire workspace.

You can group problems by problem type, which will separate, e.g. validation problems from expression errors. You can also open a problem in the Source Page.

To group problems

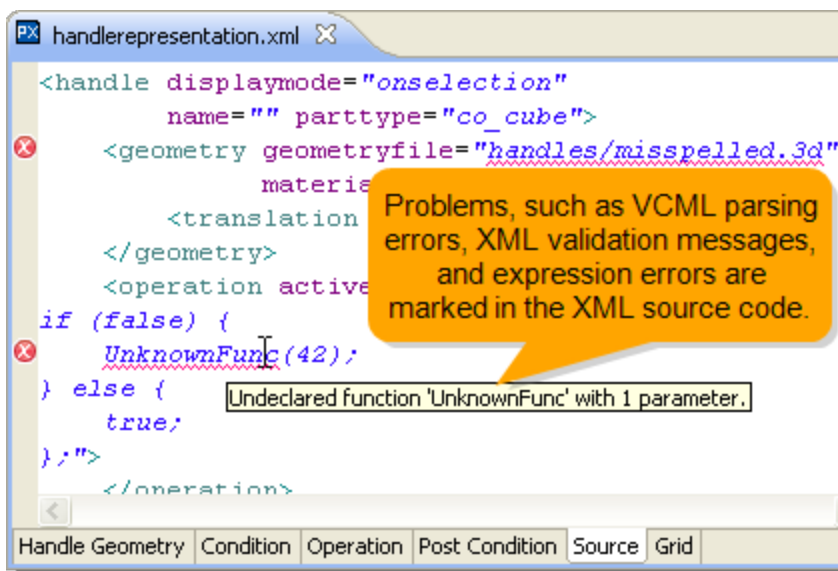
- Click the **View Menu** button in the top right corner of the **Problems View** and choose **Group By** from the context menu.



Note: If the corresponding VCML Decorator is enabled, the elements with problems are also marked in the Project Explorer.

To open a problem in the Source Page

- Double-click a problem in the **Problems View**. The Source Page will open at the location where the problem occurs.



Search View

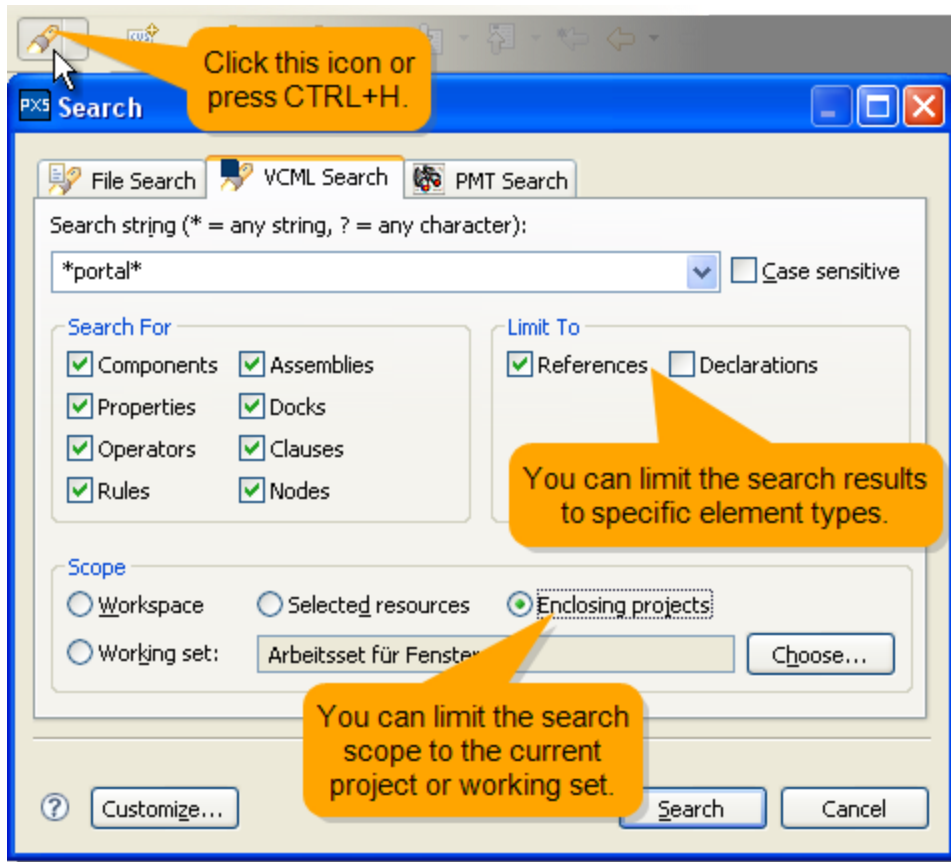
The Search View displays search results.

To search

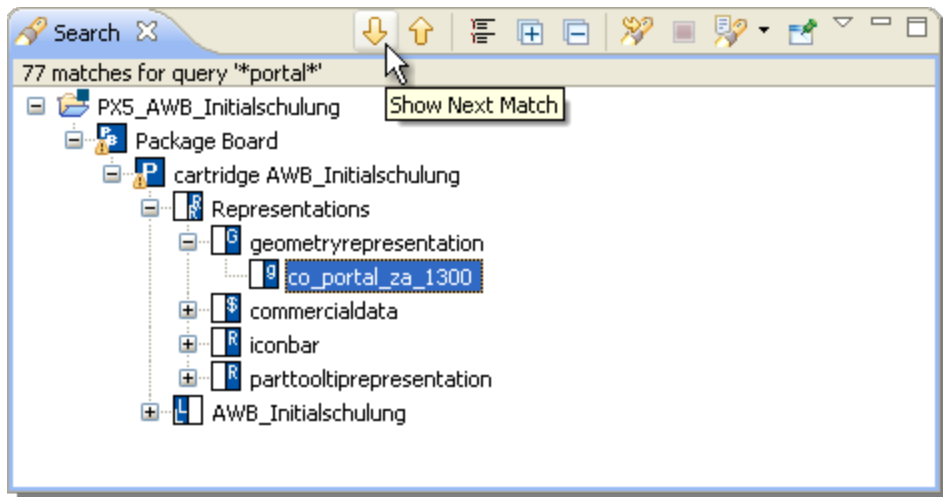
1. Click the **Search** icon in the toolbar or press CTRL+H.
2. Enter your search criteria in the **Search** window and click **Search**.

In the **VCML Search** tab you can search for components, assemblies, and other VCML elements in your project.

Note: The VCML search does not yet find references within expressions.



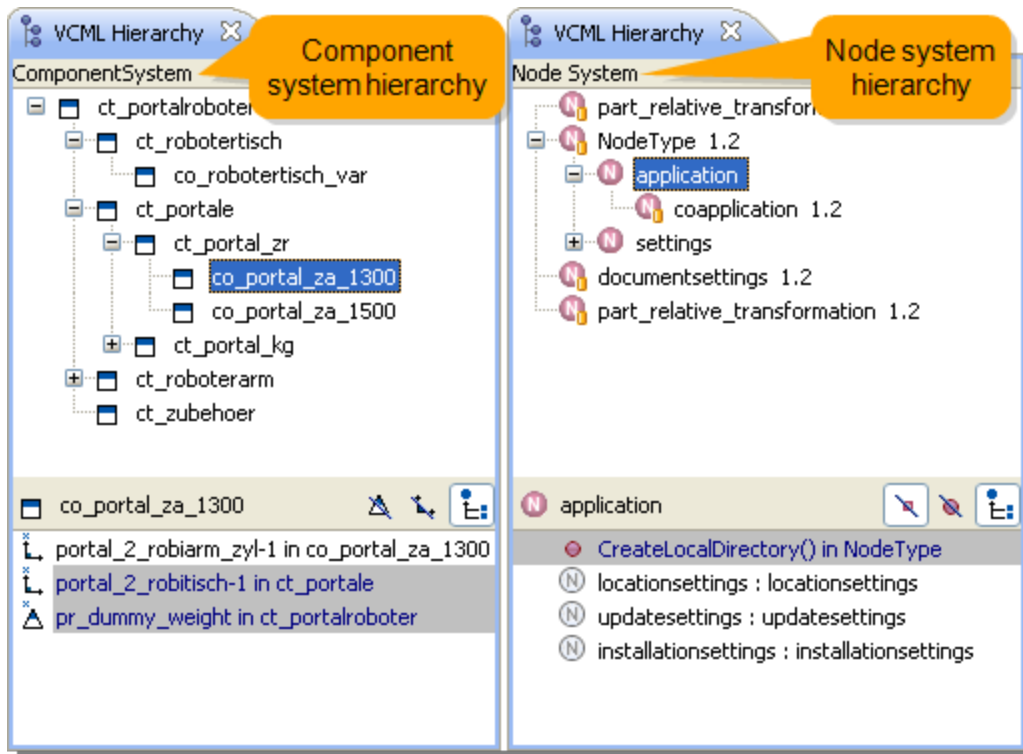
3. In the **Search View**, click the **Next** and **Previous** icons or press CTRL+PERIOD and CTRL+COMMA to navigate through the elements.



Tip: To quickly find all references to a VCML type, right-click a VCML type and choose **References** from the context menu or press CTRL+SHIFT+G.

VCML Hierarchy View

The upper pane of the VCML Hierarchy View shows the inheritance structure of components, assemblies, nodes, and other top-level VCML types. The details pane underneath, lists the members of the selected type, such as dock assignments for components or node methods for nodes.



To open the VCML Hierarchy View

Do one of the following:

- Choose **Window > Show View > VCML Hierarchy** from the main menu.
- Right-click a VCML element in the **Outline** and choose **Show In > VCML Hierarchy** from the context menu.

Polyedit

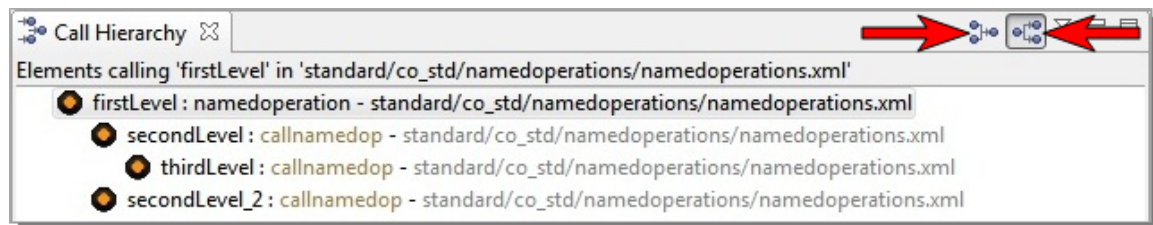
The "Polyedit" 3D Viewer is embedded in the Authoring Workbench for displaying and editing geometries of components.

You can assign multiple 2D and 3D geometries as representations to a component. Every geometry can set its materialization, positioning, scaling and activation status or have them calculated via expressions.

Call Hierarchy View

The Call Hierarchy View shows the hierarchy of named operation calls in a tree.

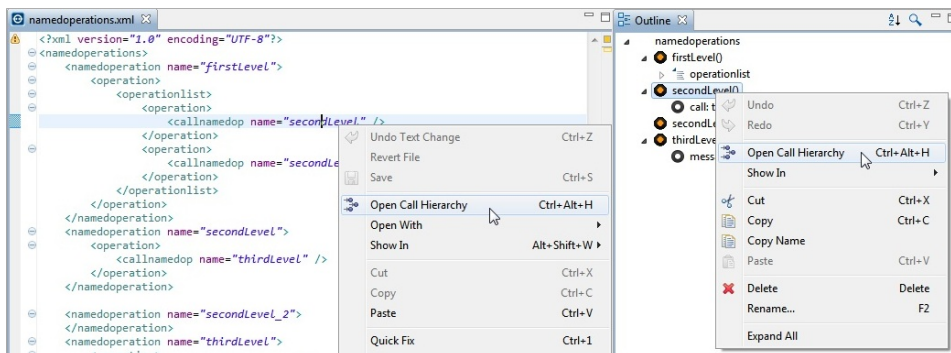
You can switch between all elements, which are calling the selected named operation (callers) or all named operations, which are called by the selected named operation (callees).



To open a named operation in the Call Hierarchy View

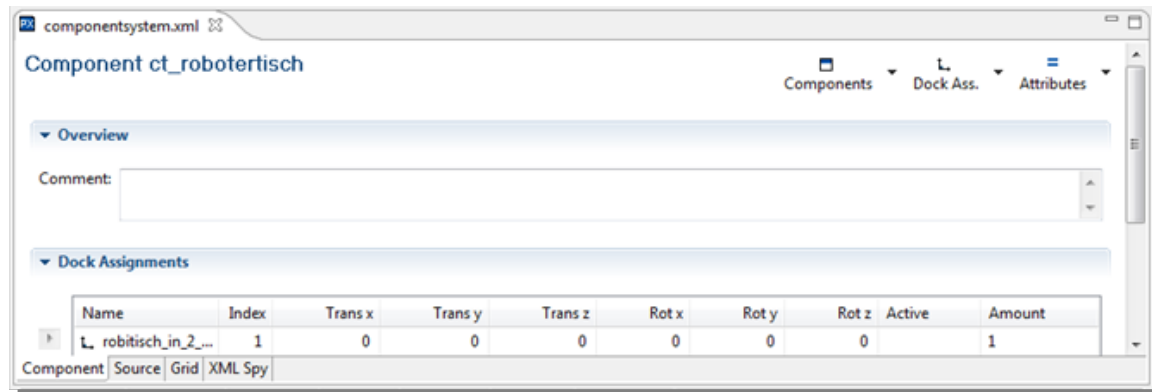
Do one of the following:

- Right-click a named operation in the named operations Source Page or in the **Outline** and choose **Open Call Hierarchy** from the context menu.
- Press CTRL+ALT+H.



Editors

The Editors contain clearly laid out fields and input assistance.

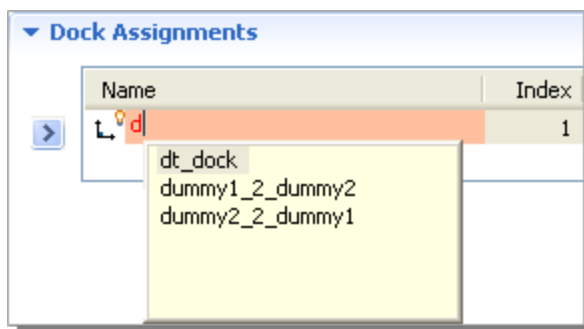


Content Assist

Whenever content assist can help you enter correct values, form elements are marked with a light bulb.

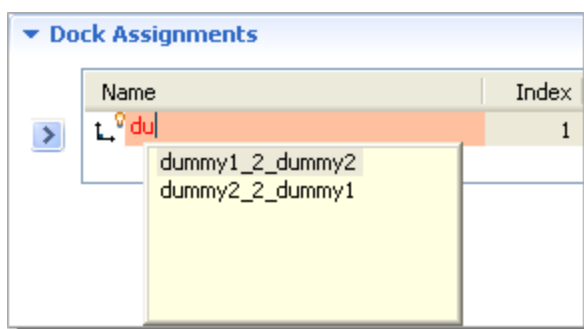
To see a list of proposals

1. Press CTRL+SPACE to see a list of proposals.

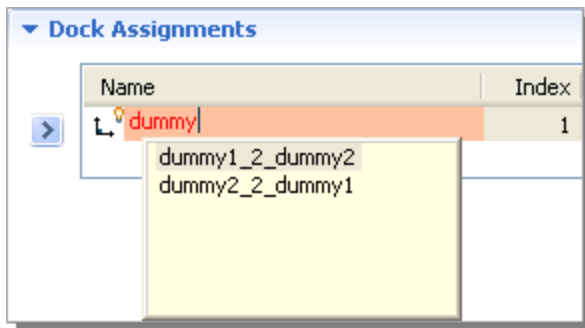


2. Do one of the following:

- Select a proposal with the arrow keys and press ENTER to apply it.
- Continue typing to narrow down the proposals (e.g. type 'u' to filter the list)...



...and press CTRL+SPACE again to complete the common prefix of all shown proposals.



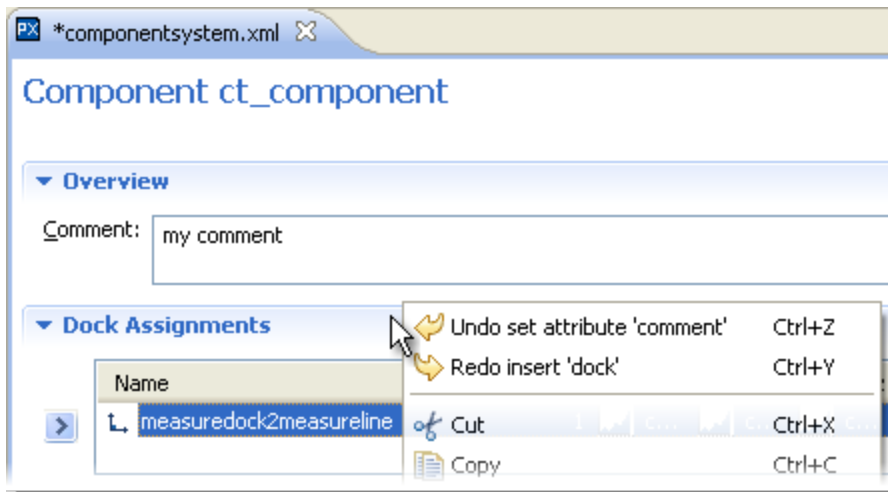
Undo/Redo Changes

You can undo any changes to the underlying XML source code performed from the VCML editors.

To undo/redo changes

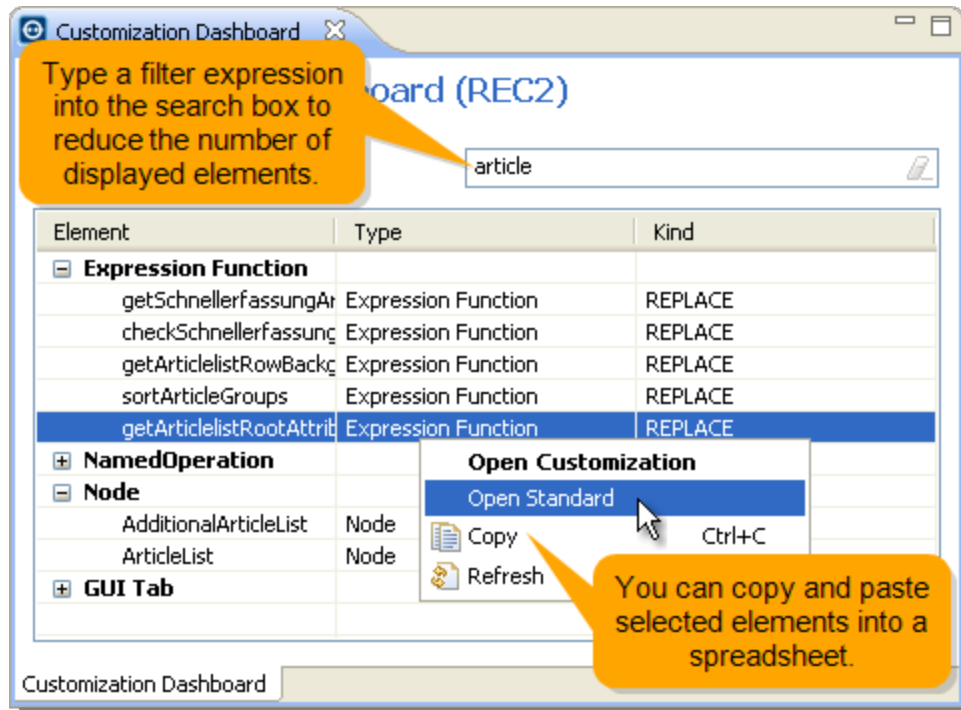
Do one of the following:

- Press CTRL+Z to undo or CTRL+Y to redo changes.
- Choose **Edit > Undo** or **Redo** from the main menu.



Customization Dashboard

The Customization Dashboard shows an overview of all elements, which have been customized (i.e. named operations, nodes, expression functions), replaced, extended, or referenced in a project.

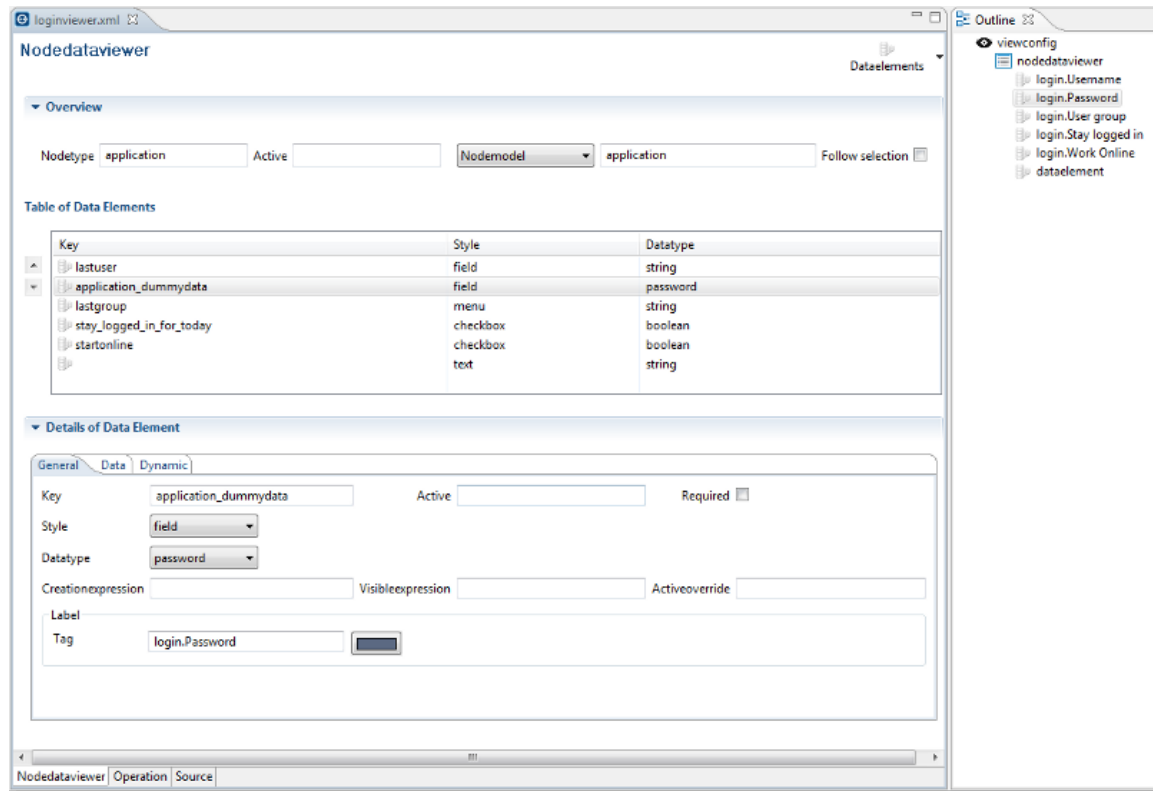


To open the Customization Dashboard

- Right-click a project in the **Project Explorer** and choose **Customization Dashboard** from the context menu.

Nodedataviewer Editor

In the Nodedataviewer Editor you can configure the Data Viewers of your project. The Nodedataviewer Editor offers full support for all types (menus, fields, color picker, etc.).



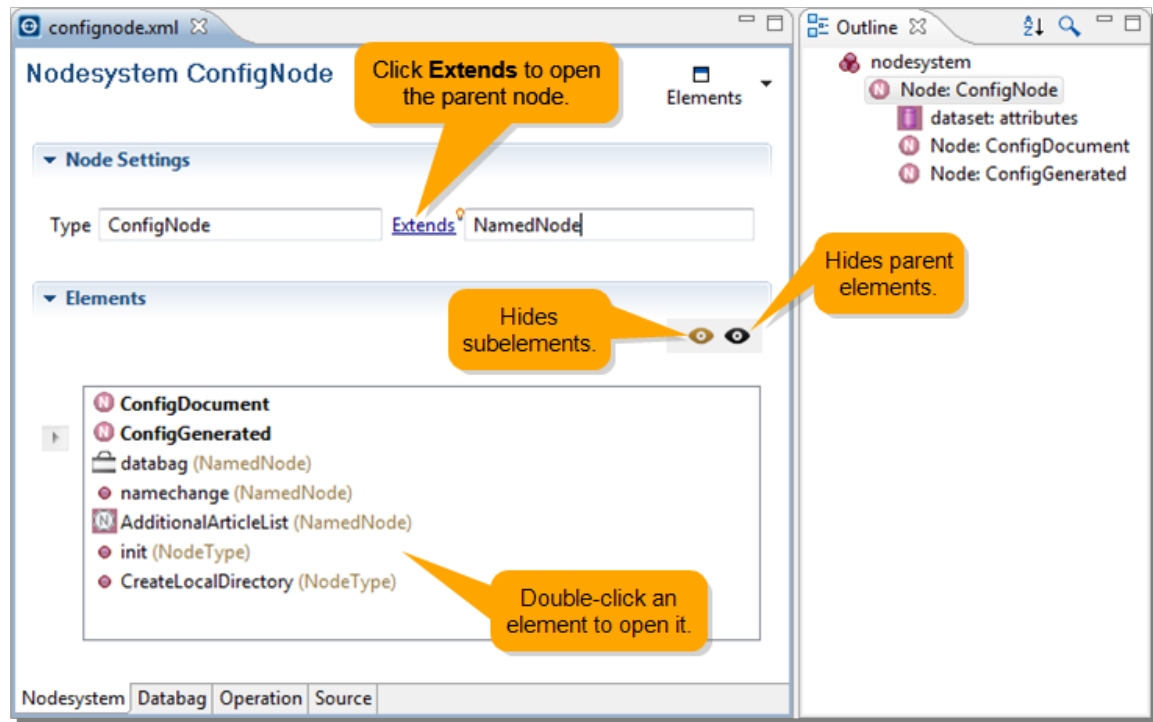
Note: The Nodedataviewer Editor only supports rasteredcolumn layouts. Deprecated layouts are not supported.

Nodesystem Editor

In the Nodesystem Editor you can edit the most important elements of the node system and you can hierarchically navigate to subelements and parent elements.

The Nodesystem Page

On the Nodesystem Page you can edit the properties of the selected node and you can open subelements and parent elements.



You can edit the name and the optional parent node. Click the Extends link to open the parent node directly in the Nodesystem Editor. That way you can navigate within the node hierarchy. You can edit the names of the child elements in the table.

Double-click an element in the table to open it. Depending on the element, either the editor or the page opens, e.g. clicking a method element opens the Operation Page of the corresponding node and selects the chosen Operation.

The Databag Page

On the Databag Page you can add, edit and delete databags.

The screenshot displays the 'Databag' page within a software application. The main window features a table titled 'Data Item' with the following columns: Key, Default, Modifia..., No dep..., Name Tag, and Category.... The table lists various keys and their corresponding default values, along with checkboxes for modification and dependencies.

Key	Default	Modifia...	No dep...	Name Tag	Category...
fullname	NodeValue(node, 'description') ++ NodeVa...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	fullname	fullname
conflict_level	MaxNodeValueFromList(ChildNodeList(no...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
technical_appr...	MaxNodeValueFromList(ChildNodeList(no...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
nodename	Localize('unnamed');	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
deliverydate_f...	FormatDate(NodeValue('deliverydate'), 'yyy...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
description		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
positionNumb...	call getPositionNumber();	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
fullPositionNu...	node ne RootNode() ? NodeValue(RootNo...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
total_price_unit	NodeValue('calculated') ? ListSum(NodeV...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
total_cost_unit	my datafield = 'total_cost'; NodeValue('cal...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
total_net-price...	my datafield = 'total_net-price'; NodeValue...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
total_weight_u...	my datafield = 'total_weight'; NodeValue('c...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
total_price	NodeValue('totalquantity') * NodeValue('to...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
total_cost	NodeValue('totalquantity') * NodeValue('to...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
total_net-price	NodeValue('totalquantity') * NodeValue('to...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
total_weight	NodeValue('totalquantity') * NodeValue('to...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
calculated	node ne RootNode() ? NodeValue(ParentN...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
quantity	1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
totalquantity	node ne RootNode() ? NodeValue(ParentN...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		

On the right side, the 'Outline' pane shows a tree structure of nodes:

- nodesystem
 - Node: DocumentNode
 - method: init
 - Node: NamedNode
 - databag
 - method: namechange
 - AdditionalArticleList
 - Node: Folder
 - Node: AdditionalArticleList
 - Node: addresses
 - generic_address
 - generic_address
 - generic_address
 - Node: user_items
 - databag
 - Node: generic_address
 - databag
 - method: address_check_ar
 - method: modified_addres
 - Node: costs
 - databag

The Operation Page

On the Operation Page you can edit method elements of the node system.

The screenshot displays the 'Operation of method "address_check_and_update"' page. It features a table of operations, an 'Operation Editor' for the selected 'operationlist', and an 'Outline' panel on the right.

Operation	Comment	Wait	Active / Marked parts	Structure
operationlist		0	/*Ä¶We have 2 case to u...	
message		0		
setvalues		0		
source				
target				
nodemethod		0		

Operation Editor

Properties of Operation "operationlist"

Returnvariable: Contextviewer:

Executionorder: Showprogress: ☐

Progressticks: Progresstext:

SimpleOp Editor

Condition

Nodesystem Databag Operation Source

Outline

- nodesystem
 - Node: DocumentNode
 - method: init
 - Node: NamedNode
 - databag
 - method: namechange
 - AdditionalArticleList
 - Node: Folder
 - Node: AdditionalArticleList
 - Node: addresses
 - generic_address
 - generic_address
 - generic_address
 - Node: user_items
 - databag
 - Node: generic_address
 - databag
 - method: address_check_and_update
 - operationlist
 - message
 - setvalues
 - source
 - target
 - nodemethod
 - method: modified_address
 - Node: costs
 - databag

Rule Editor

The Rule Editor allows the intuitive modeling of product knowledge. The display of the visual rules links the product structure and logical conditions in one integrated overview. The structural relations are displayed as symbols for aggregations (has part) and as a composition (is partner). The logical conditions are represented as symbols, which are based on the traffic light metaphor. The traffic light symbol is either red (false) or green (true), due to the rule validation in the Configurator. Structural dependencies (has part, is partner) and logical conditions (and, or, xor, not) are structured in the Rules Editor as a graphical model.

Translation Editor

In the Translation Editor you can translate the language of the user interface. The Translation Editor lists text blocks from the product information and the configuration and application logic (or its user interface) in the requested languages. The color coding shows which text elements have not been translated yet. You can export the text to Excel and then re-import it after it has been translated by third parties.

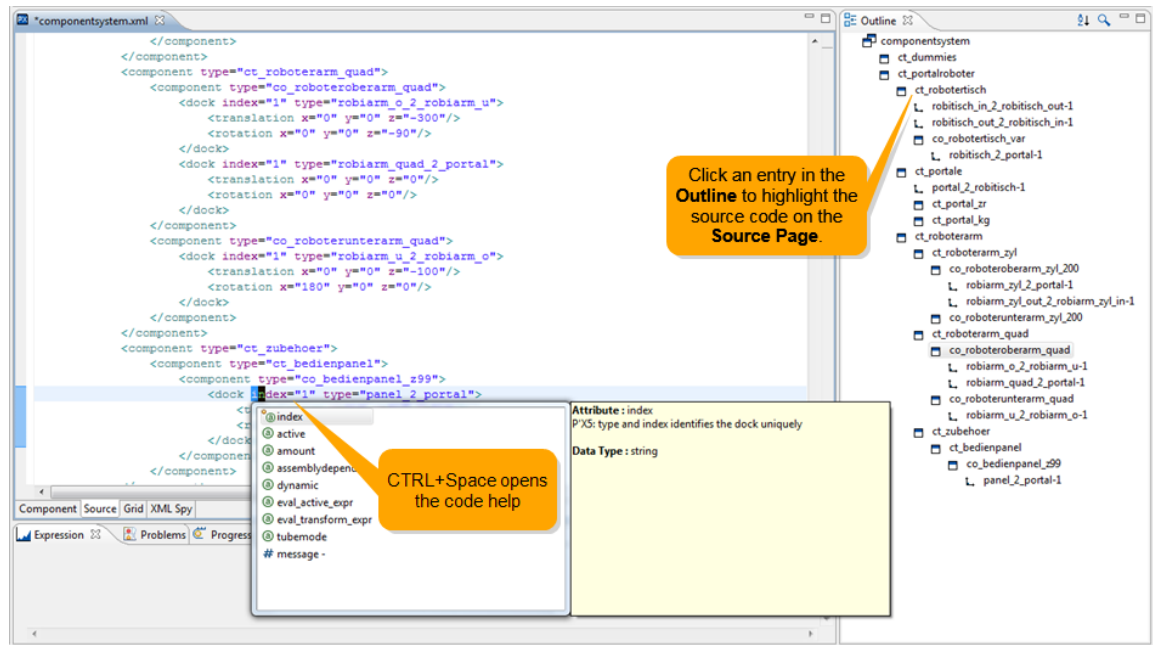
Pages

The Source Page (XML Editor)

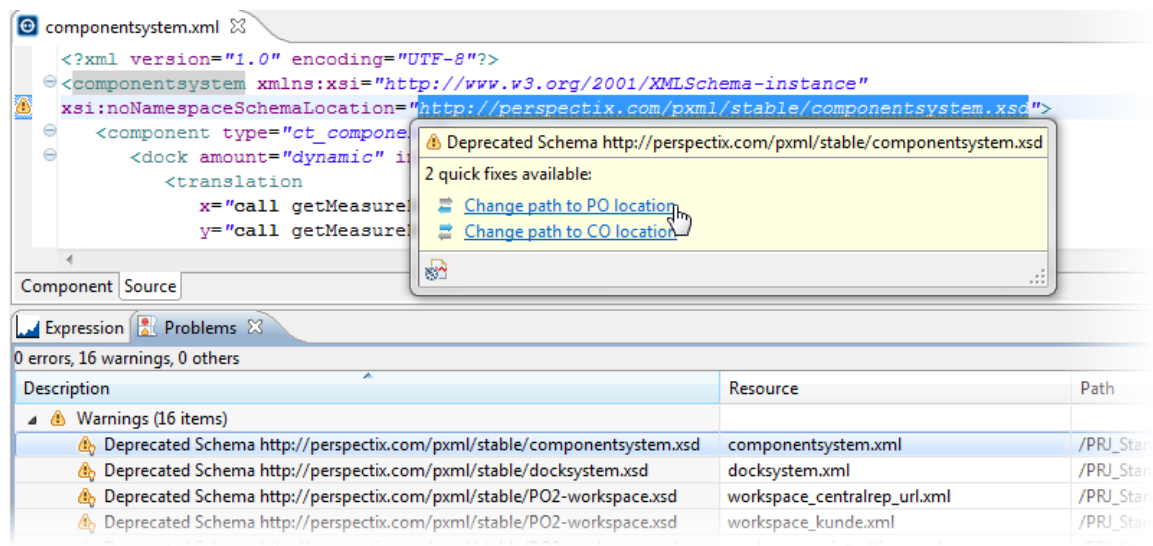
On the Source Page you can edit the XML code with the XML editor from Eclipse Web Tools Project (WTP).

The XML editor provides syntax highlighting and content-assist for closing tags, attributes and elements according to XSD, and validates the XML file according to the underlying schema.

You can turn any named element, such as component or assembly names, operator references, properties, features, node names, etc. into hyperlinks.

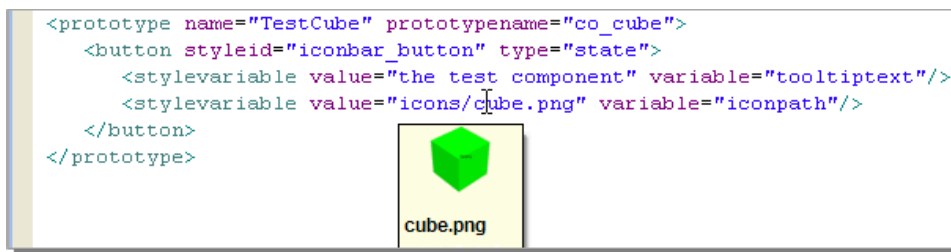


The Source Page detects deprecated or missing schemas and helps you fix them by proposing possible resolutions.



To see the image of a path to an image file

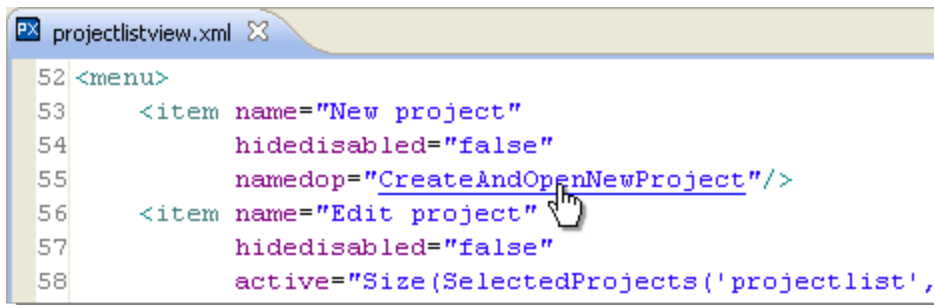
- Place the mouse cursor over the path to the image file. The image appears as a pop-up.



To turn a VCML element into a hyperlink

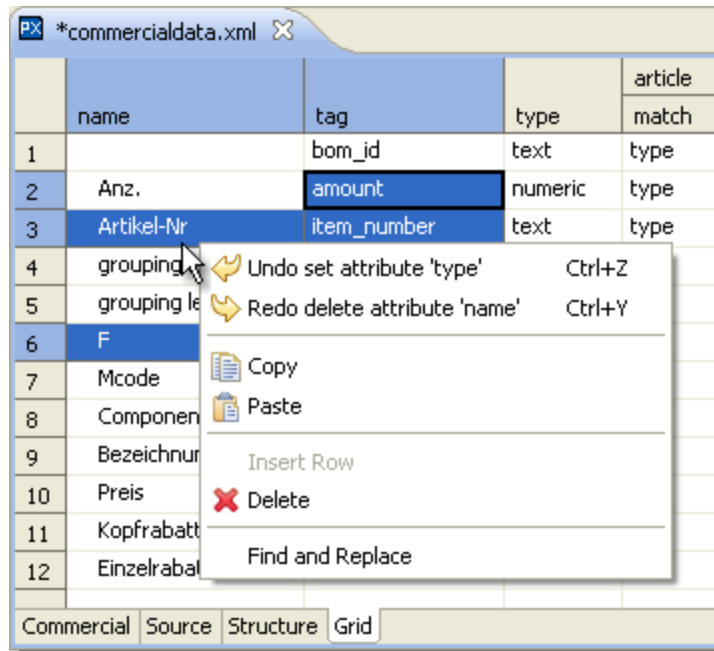
You can turn any named element, such as component or assembly names, operator references, properties, features, node names, etc. into hyperlinks.

- Press and hold down the CTRL key.



The Grid Page

On the Grid Page you can edit data in tables, e.g. materialmaster tables.



To always show the Grid Page in the Editor

1. Choose **Window > Preferences** to open the **Preferences** window.
2. Choose **VCML > Editor** in the **Preferences** window and select the **Table** checkbox.

The Dock DOF Page

On the Dock DOF Page in the Componentsystem Editor you can overwrite DOF definitions in the dock system.

componentsystem.xml

Component co_table_var

Dock Ass.

▼ Dock Assignments

Name	Index	Trans x	Trans y	Trans z	Rot x	Rot y	Rot z	Active	Amount
↳ table_2_ga...	1	0	0	0	0	0	0		1
↳ table_out_...	1	1000	0	0	0	0	0		1

▼ Degree Of Freedom

DOF Type: rotation DOF
DOF Axis: x

▼ DOF Domain

Domain Type: step
From: 1
To: 10
Step:

☒ Numeric Values

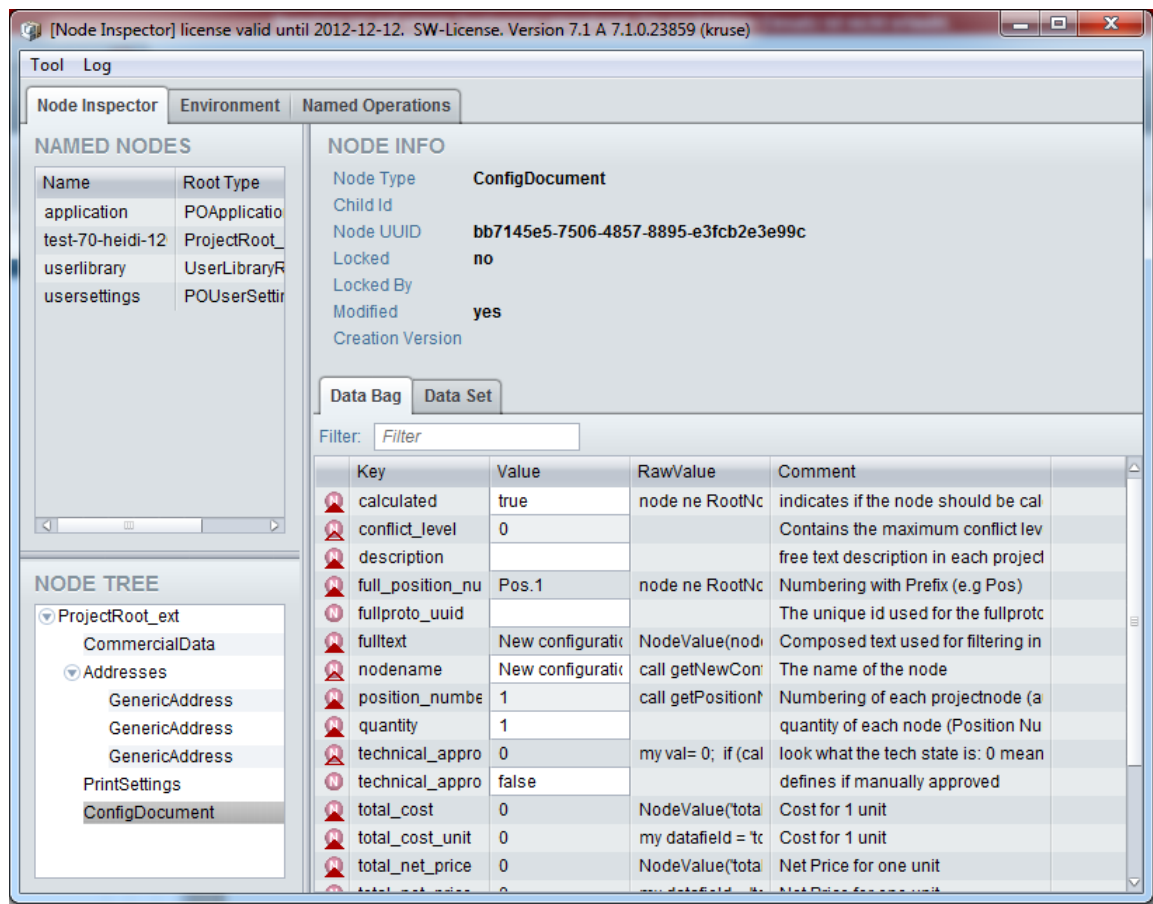
▶ Second Degree Of Freedom

▶ Second DOF Domain

Component
Dock DOF
Source

Node Inspector

The Node Inspector shows information about the currently instantiated nodes, including general node information, current databag values, and node dataset content. A context menu on the node type allows you to jump into the node definition in the Authoring Workbench.

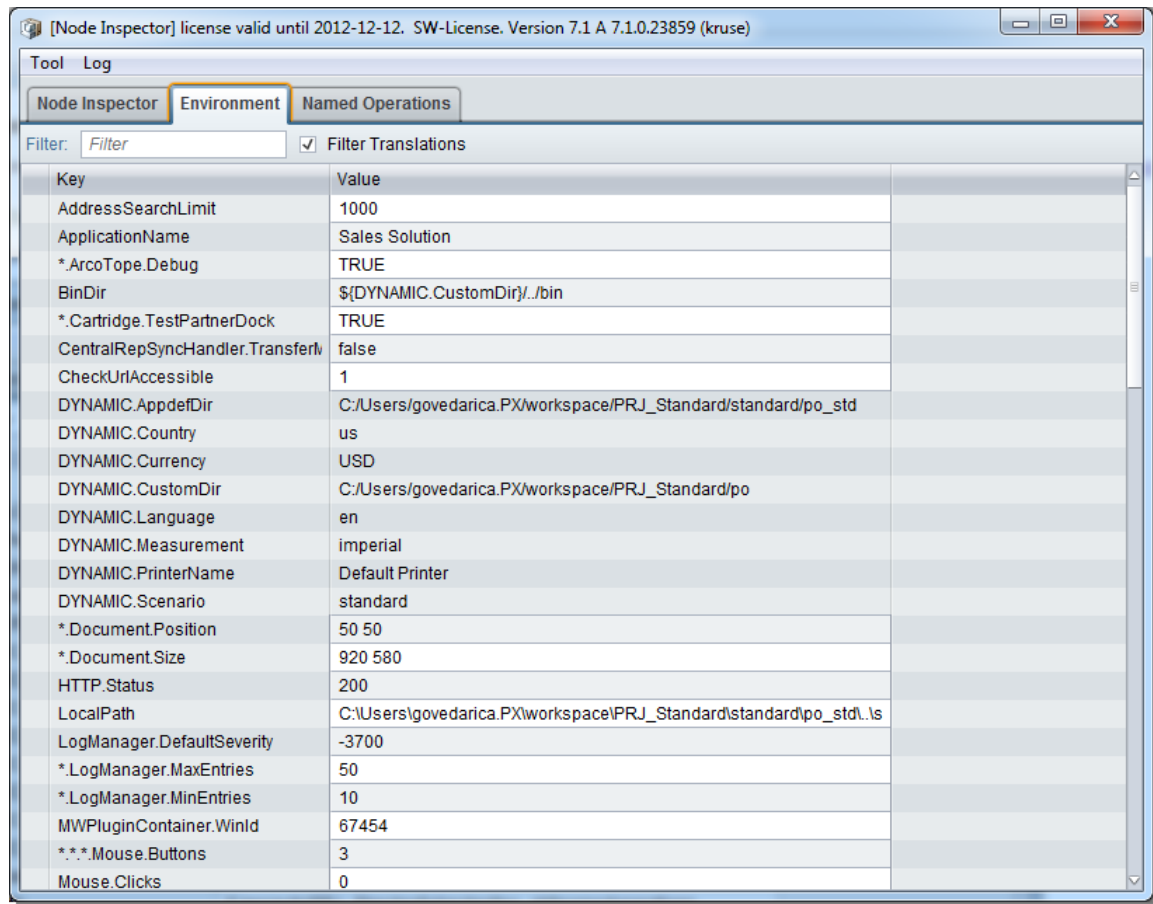


To open the Node Inspector

- Press SHIFT+F12 in the **Project Organizer** or the **Configurator** and click the **Node Inspector** tab.

Environment Variable Inspector

In the Environment tab (i.e. Environment Variable Inspector) of the Node Inspector you can browse and edit the values of all currently active environment variables.



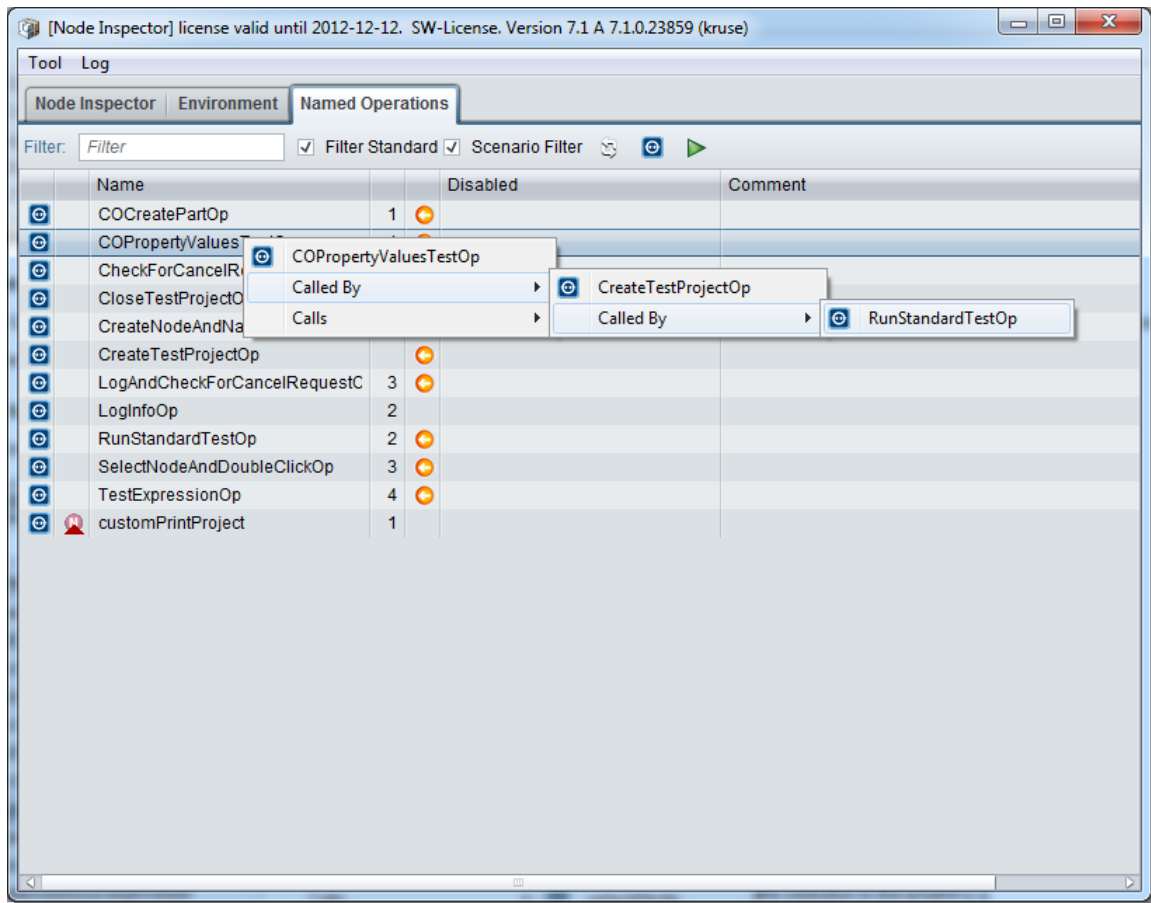
To open the Environment Variable Inspector

- Press SHIFT+F12 in the **Project Organizer** or the **Configurator** and click the **Environment** tab.

Named Operations Inspector

In the Named Operations tab (i.e. Named Operations Inspector) of the Node Inspector you can see and execute loaded named operations and enter any necessary parameters.

The context menu shows both calling and called methods, for the author to better understand the application flow.



To open the Named Operations Inspector

- Press SHIFT+F12 in the **Project Organizer** or the **Configurator** and click the **Named Operations** tab.

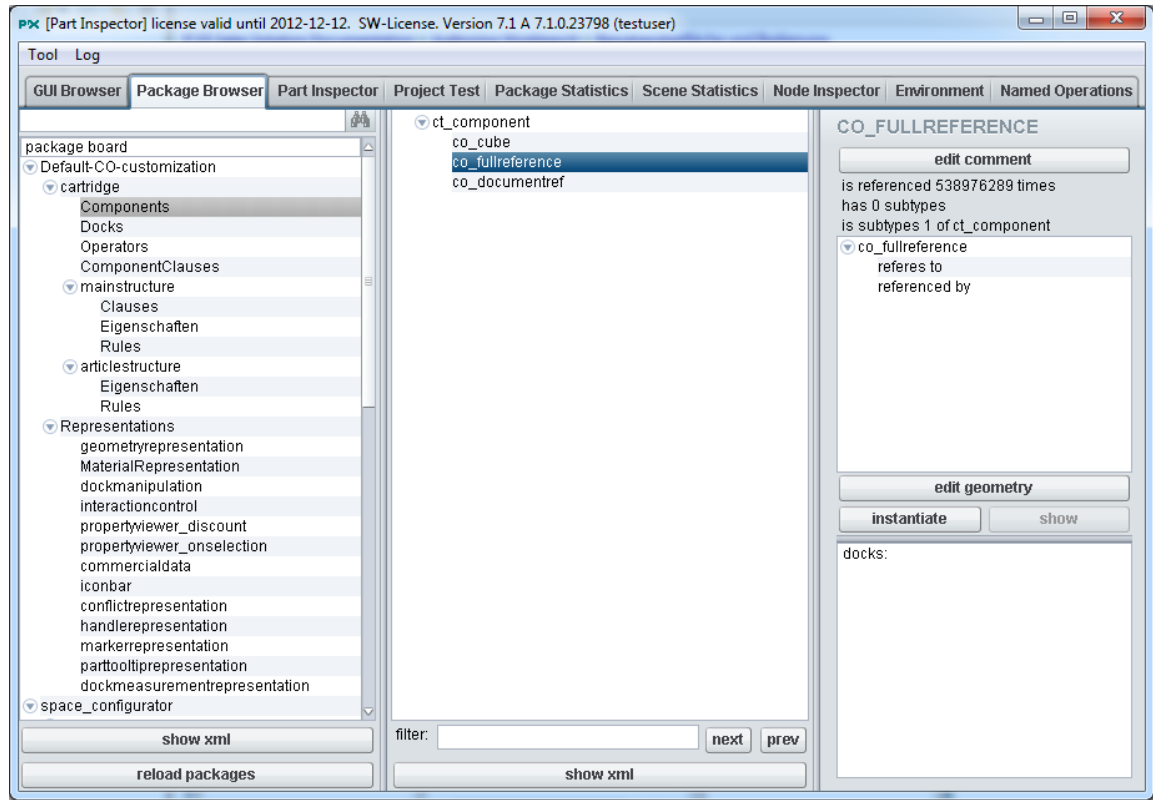
Inspector

Press SHIFT+F12 in the Configurator to open the Inspector.

The Package Browser Tab

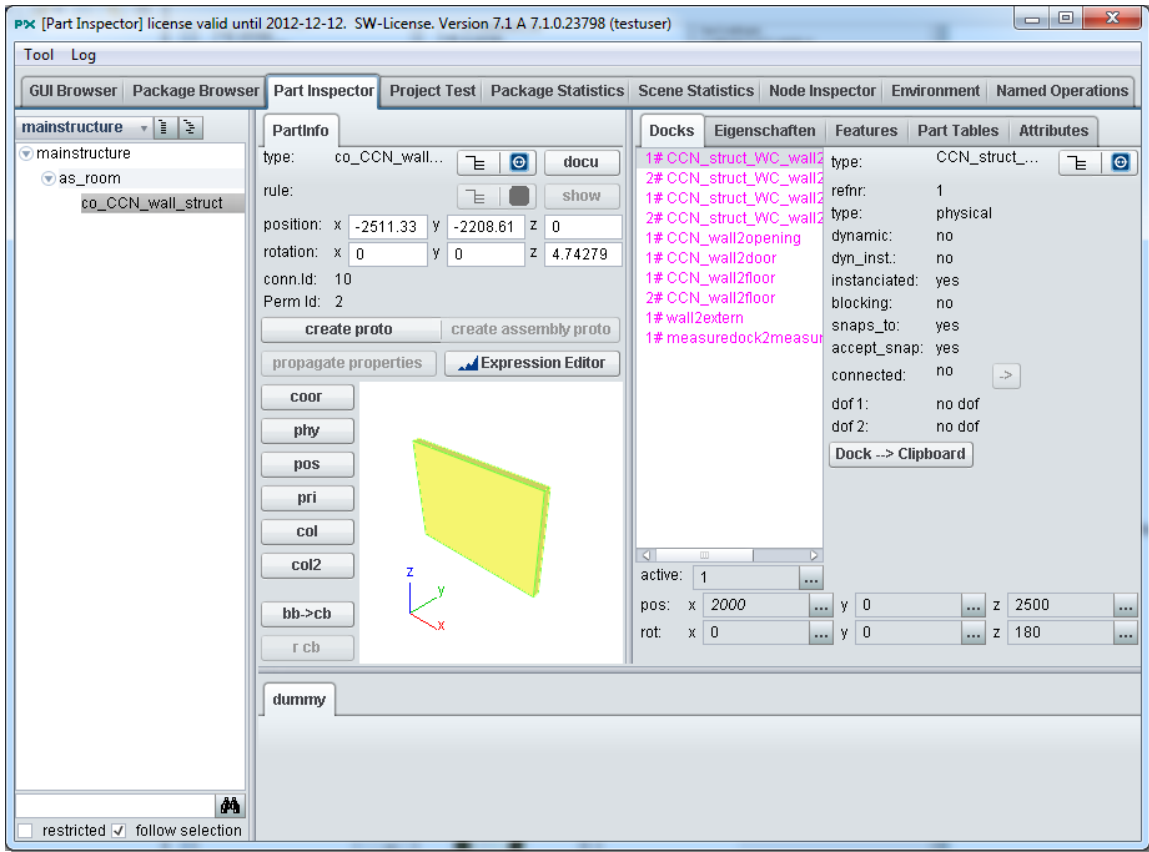
The Package Browser tab in the Inspector:

- Contains an overview of all structures (systems) in the cartridge (meta information level).
- Serves as access to the XML editor to specifically edit individual components, rules, etc.
- Provides search functions to find individual components, rules, etc. in the files.



The Part Inspector Tab

The Part Inspector tab in the Inspector shows Information from the instance world and allows the targeted viewing and debugging of conditions, dock positions, and properties.



With the buttons **create proto** or **create assembly proto** the generated assembly prototypes have the extension pxaz.

- **Create proto** opens the file browser, so you can choose the location for saving the protptype file.
- **Create assembly proto** automatically stores the prototype file inside the folder **parts** inside the corresponding cartridge of your project Use this only when you start the application from inside the awb.

Preferences

In the Preferences window you can define your personal settings. You can also import and export your preferences. See page 155.

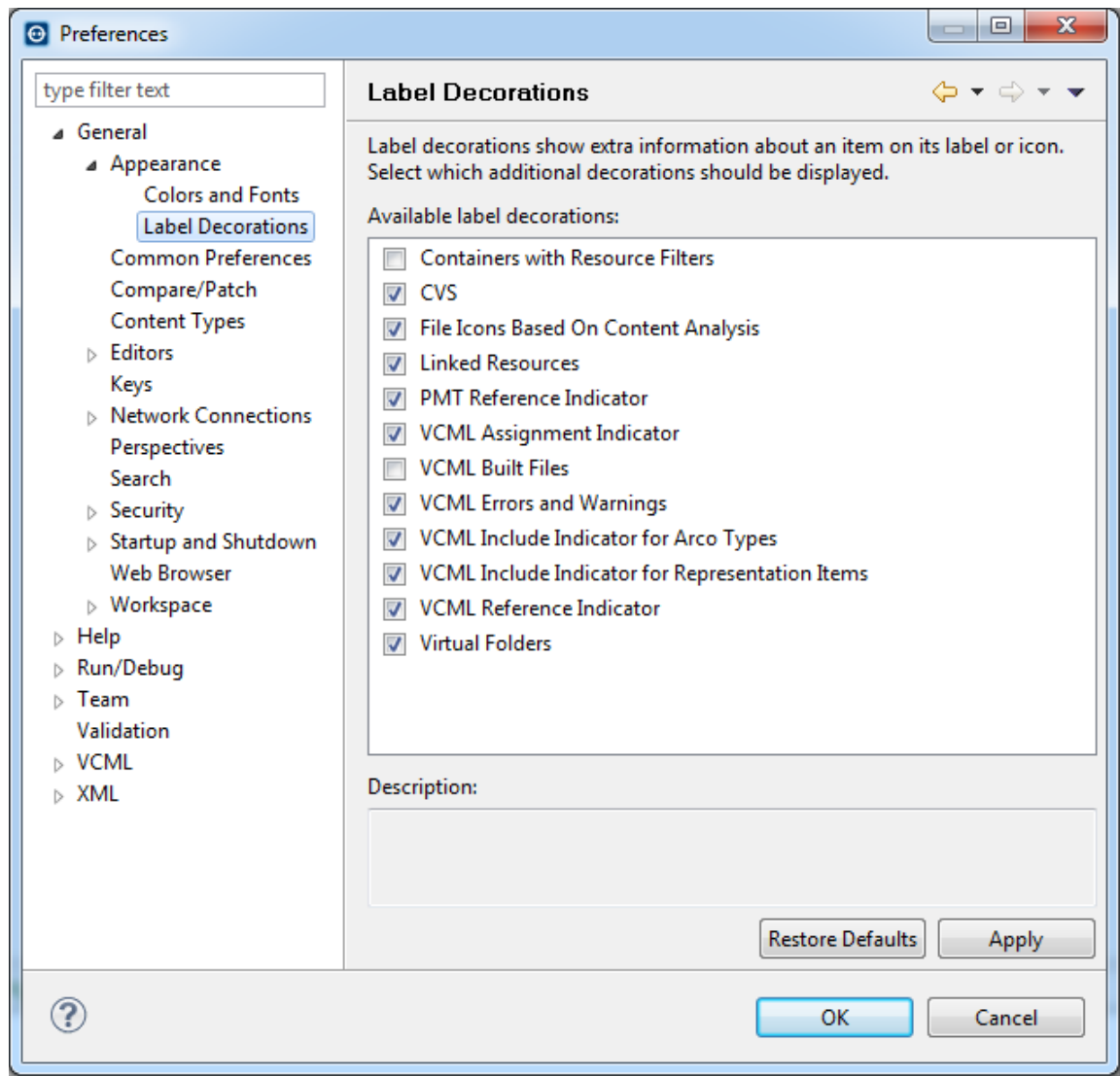
To open the Preferences window

- Choose **Window > Preferences** from the main menu.

Label Decorations

On the Label Decorations page you can define which additional information is displayed on the label or icon of an item.

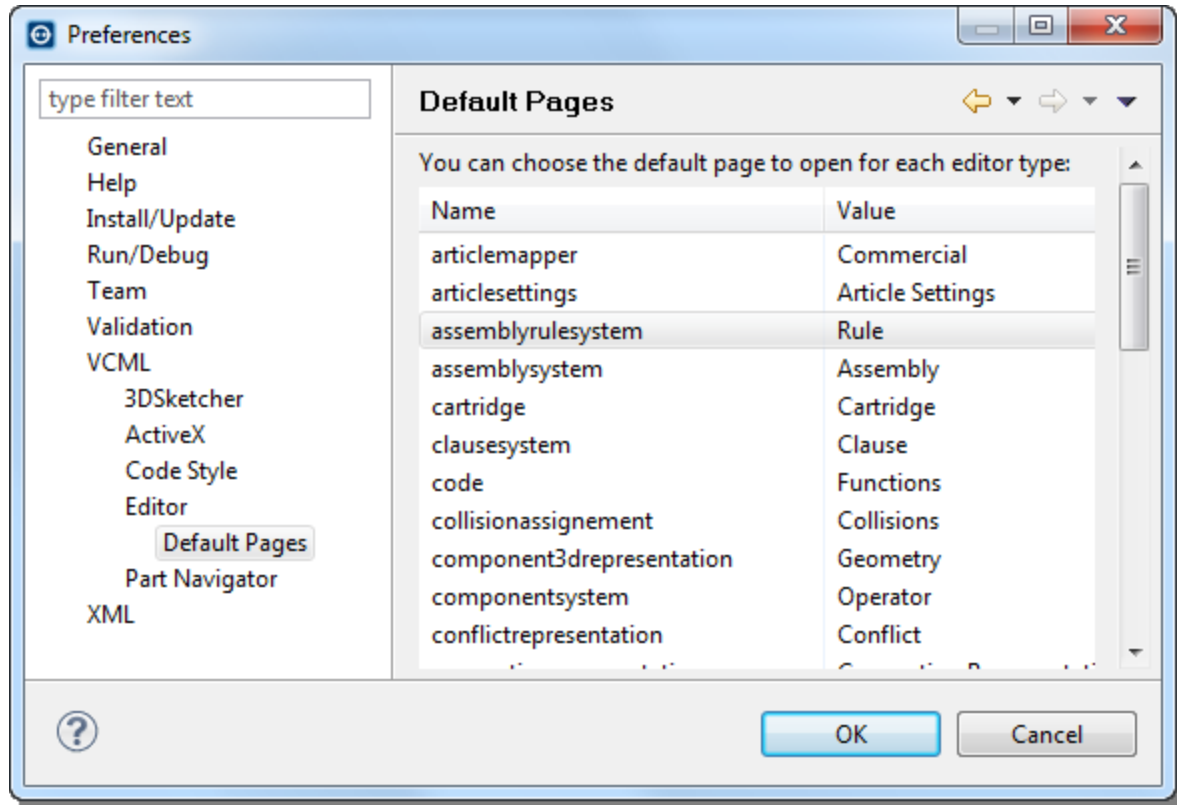
Choose **General > Appearance > Label Decorations** in the **Preferences** window.



Default Pages

On the Default Pages page you can define which page opens by default when you open the VCML Editor.

Choose **VCML > Editor > Default Pages** in the **Preferences** window.

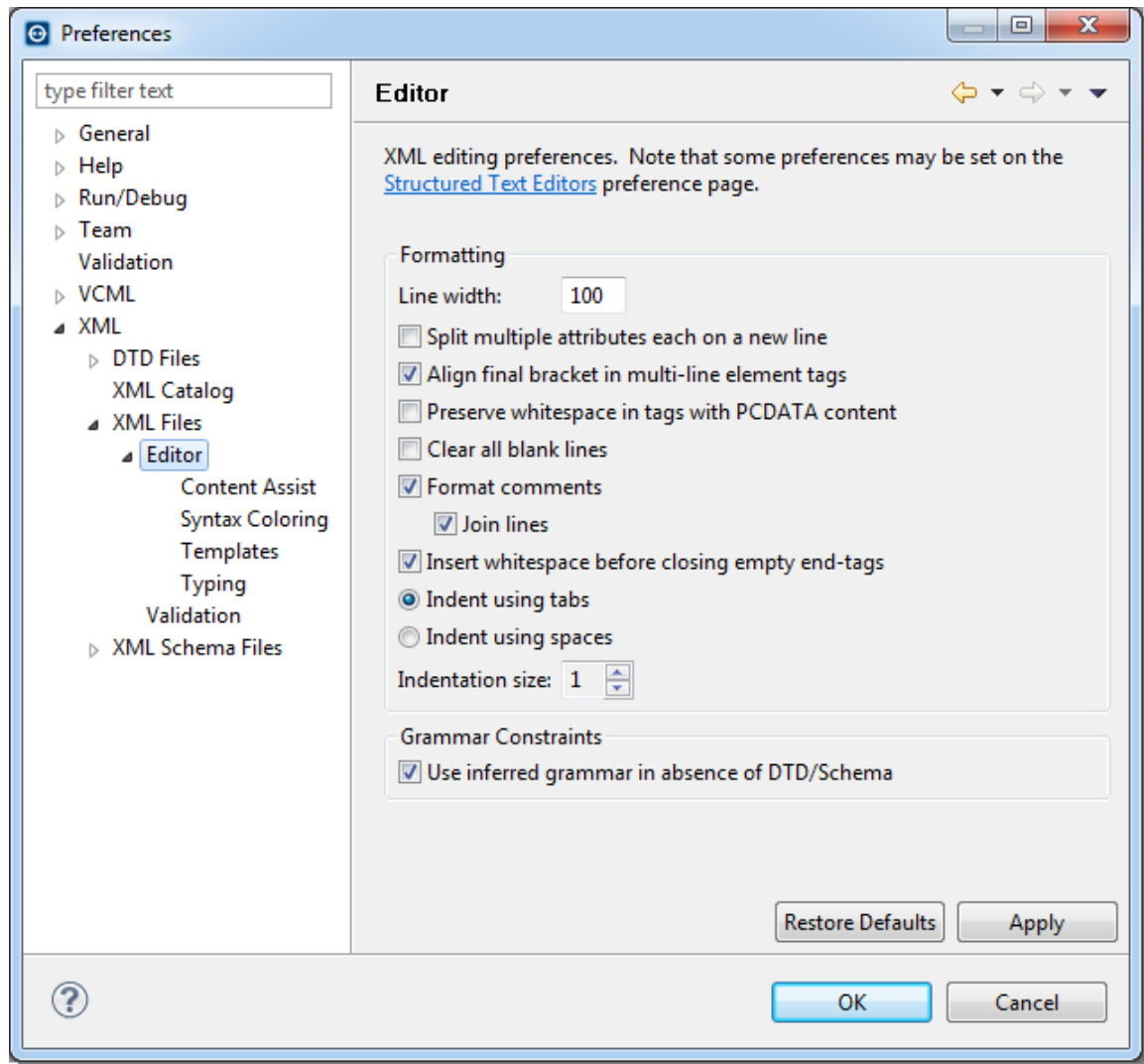


Version: Requires P'X5 version 6.2 or higher.

Editor

On the Editor page you can define your XML formatting preferences.

Choose **XML > XML Files > Editor** in the **Preferences** window.

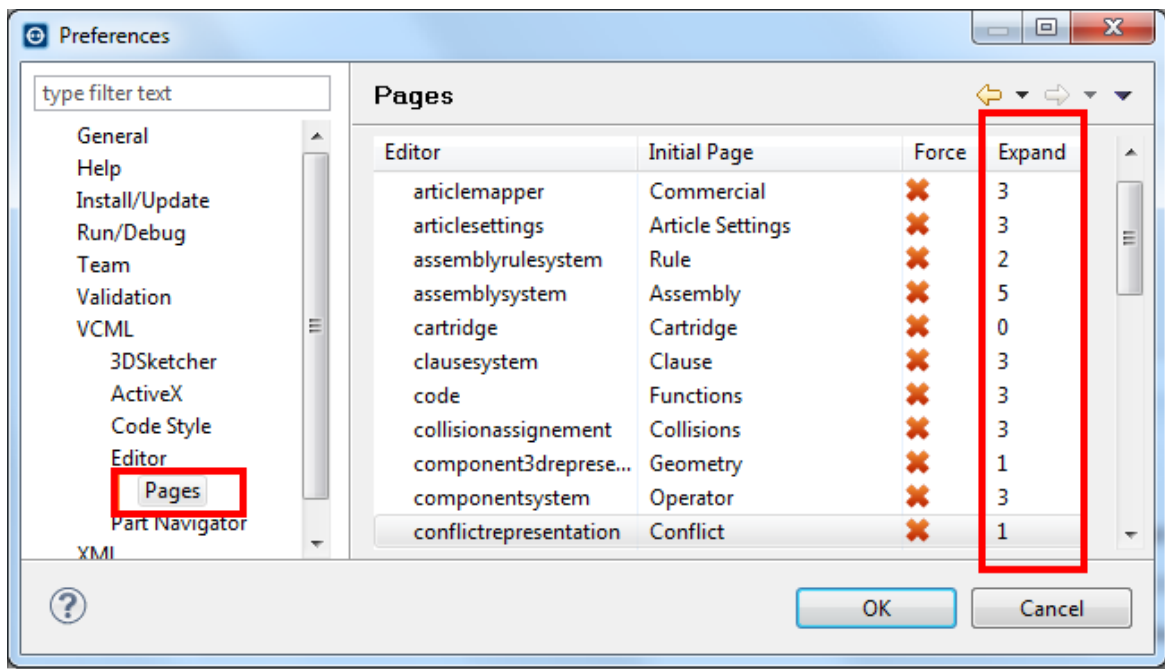


Note: The Authoring Workbench editor only updates the XML code that has been changed, not the entire file. This makes working together easier, since there are less cases where simultaneous changes of the file must be manually merged.

Pages

On the Pages page you can define the auto-expand level of the outline. You can configure the level for every page type (e.g. Geometry Representation) individually.

Choose **VCML > Editor > Pages** in the **Preferences** window.



Tutorials

After you complete the following tutorials, you will gain an overview of the tasks in the P'X5 Sales Solution, you will be familiar with product analysis and you will know how to work with the Authoring Workbench.

- **Basic Tutorial (Practice)** - The Basic Tutorial (Practice) teaches you step-by-step how to program the Configurator.
- **Cheat Sheets** - The Cheat Sheets are a set of tutorials closely modelled after the basic tutorial.

To download the tutorial project (PX5_AWB_Tutorial) in the Authoring Workbench

Do one of the following:

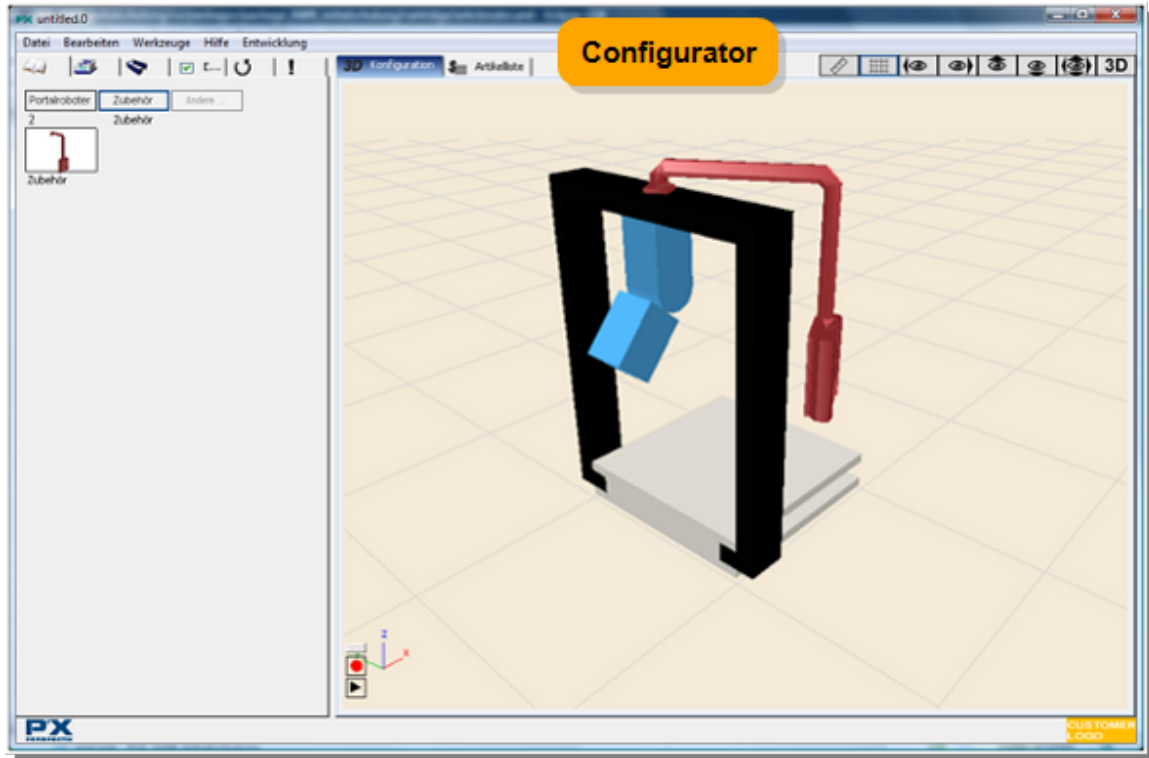
- Choose **File > New > Example** from the main menu. Choose **VCML > Tutorial Project** in the **New Example** window and click **Next**. Enter the data and click **Finish**.
- Choose **File > New > Project** from the main menu. Choose **Examples > VCML > Tutorial Project** in the **New Project** window and click **Next**. Enter your data and click **Finish**.
- Choose **Help > Cheat Sheets** from the main menu. Choose **P'X5 Authoring Workbench Tutorials > Create Example Project** in the **Cheat Sheet Selection** window and click **OK**. Follow the instructions in the **Cheat Sheet**.
- On the welcome page (which appears when you open a new **Workspace** or when you choose **Help > Welcome** from the main menu), choose the **Tutorials** link and the **Create Example Project** lesson.

Note: The tutorial project (PX5_AWB_Tutorial) does not have to be checked in on a server (CVS). It is a stand-alone project.

Basic Tutorial (Practice)

The Basic Tutorial (Practice) teaches you step-by-step how to program the Configurator.

Using a simple example, you will create parts for a robot in the Authoring Workbench. And then you will assemble the robot part by part in the Configurator.



You don't need any programming skills, everything you need you will learn in the following lessons.

Lesson 1 - Create Components

First, you must create components. Components in the Componentsystem Editor are divided into Supertypes and Types. Supertypes and Types can be hierarchically ordered in a tree structure, whereby the Supertype ranks higher than the Type.

- Supertype: `ct_robotarm_quad`
 - Type: `co_upper_robotarm_quad`
 - Type: `co_lower_robotarm_quad`
- Supertype: `ct_control_panel`
 - Type: `co_control_panel_99`

The `ct_` and `co_` prefixes are used for naming the hierarchical levels. The Supertypes receive the `ct_` (component type) prefix and the Leaf Types receive the `co_` (component) prefix. This way you can differentiate the Supertypes and the Leaf Types in the code. In the future, we will simply call them component types and components.

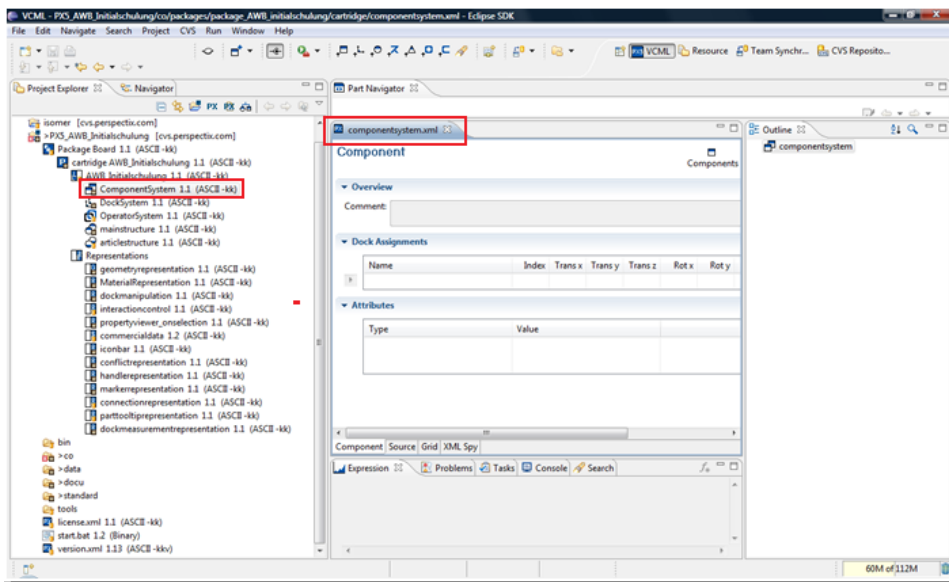
Your Assignment

Create the following components:

- Upper square robot arm
- Lower square robot arm
- Control panel 99

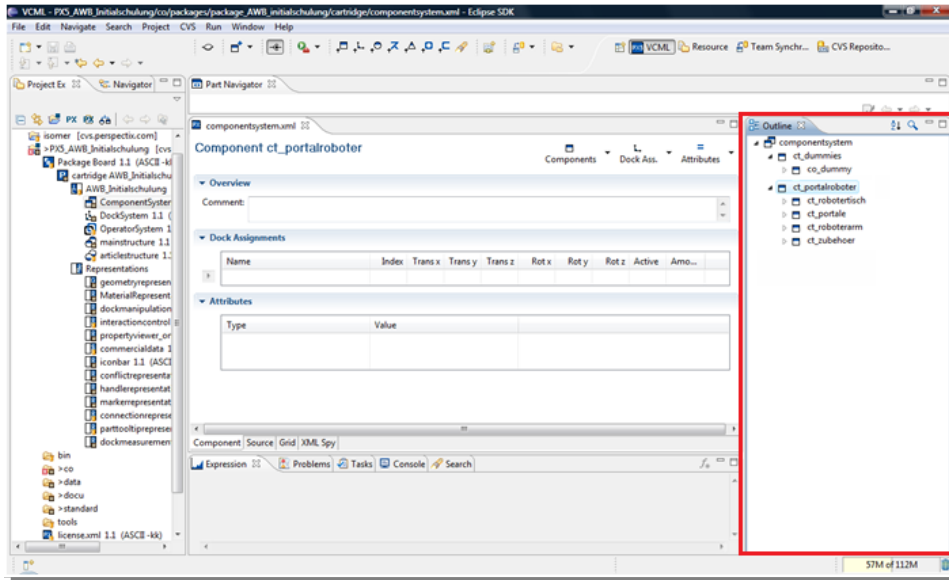
Procedure

1. Open the **Componentsystem Editor**.



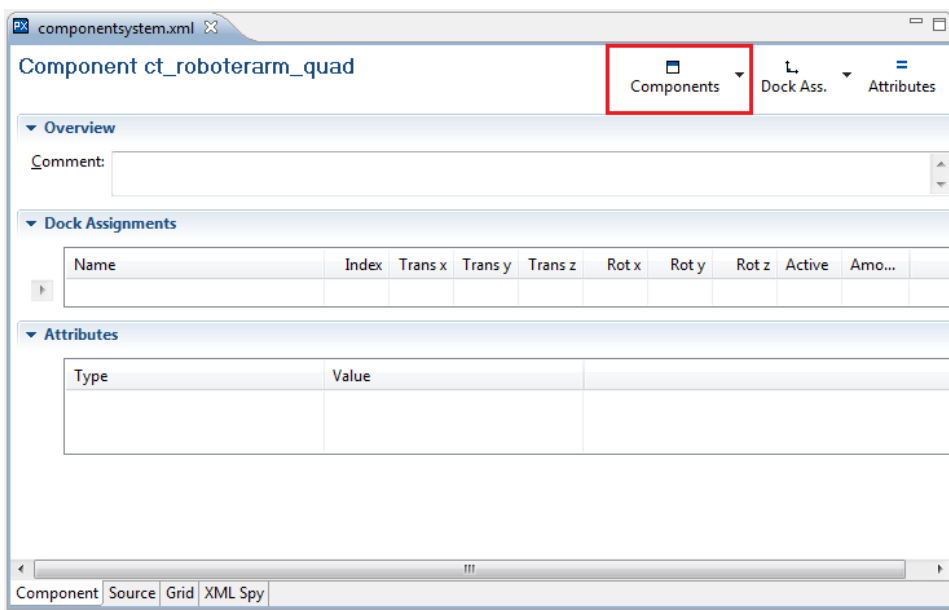
2. Take a look at the existing components in the **Outline** of the **Componentsystem Editor**. Look for the `ct_robotarm` component type and click the arrow to the left of the entry to open the hierarchy levels of the component type. The arrow appears when you place the mouse cursor over the entry.

The `ct_robotarm_cyl` component type is located underneath the component type.



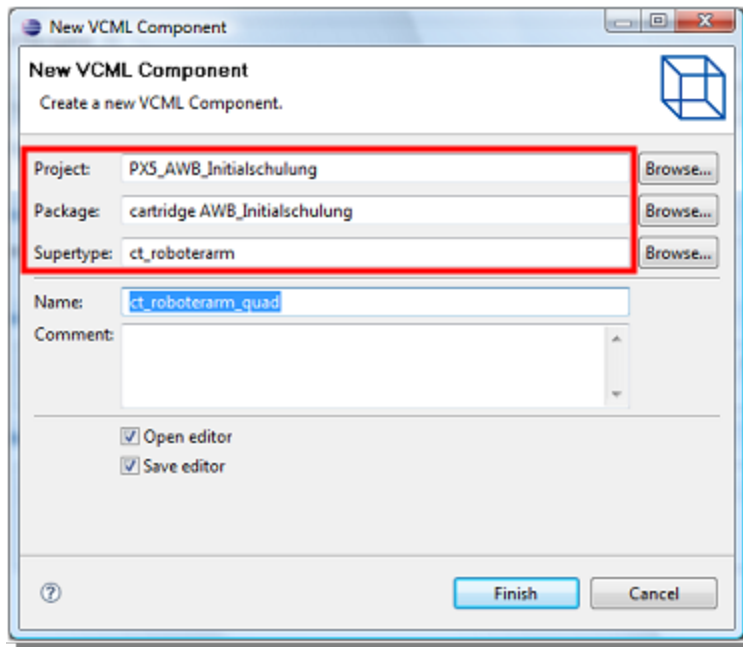
3. Create the `ct_robotarm_quad` component type underneath the `ct_robotarm` component type, just like you did with the `ct_robotarm_cyl` component type. To do this, select the `ct_robotarm` component type in the **Outline** and click the **Components** button in the **Componentsystem Editor**.

Note: The `_cyl` and `_quad` suffixes refer to the shape of the robot arm: cylindric (round) and quadratic (square).

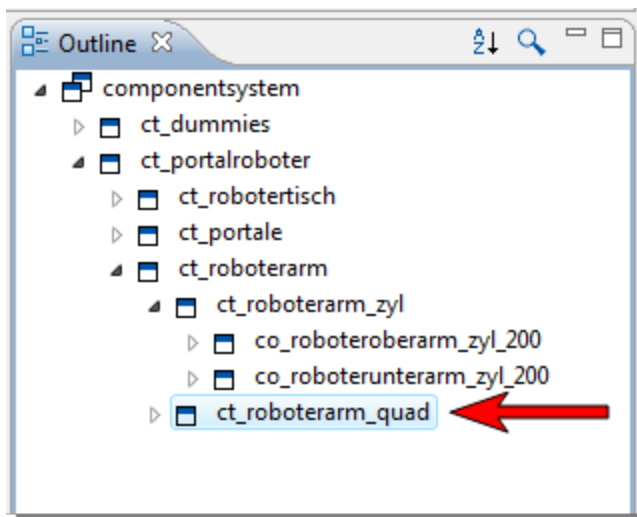


4. Type the name of the `ct_robotarm_quad` component type in the **Name** field in the **New VCML Component** window and click **Finish**.

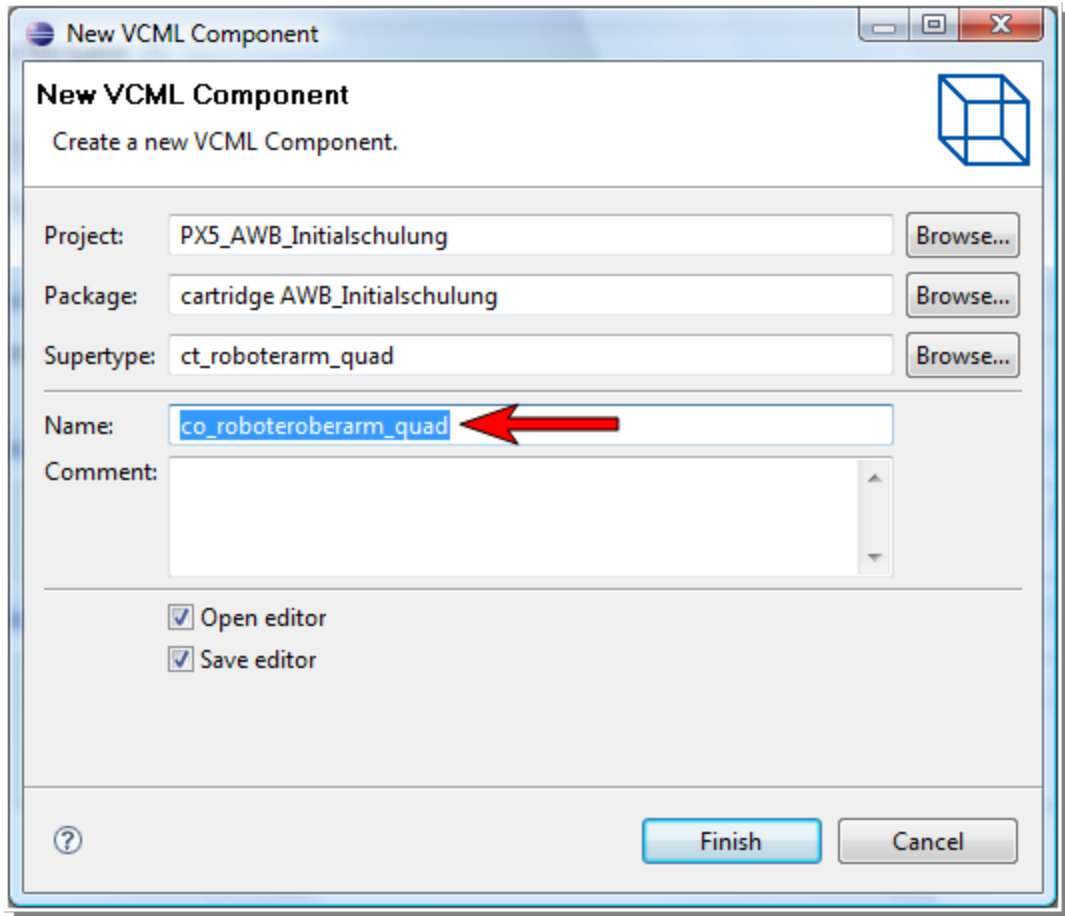
The **Project**, **Package** and **Supertype** fields already contain entries.



The `ct_robotarm_quad` component type is created in the **Outline**.



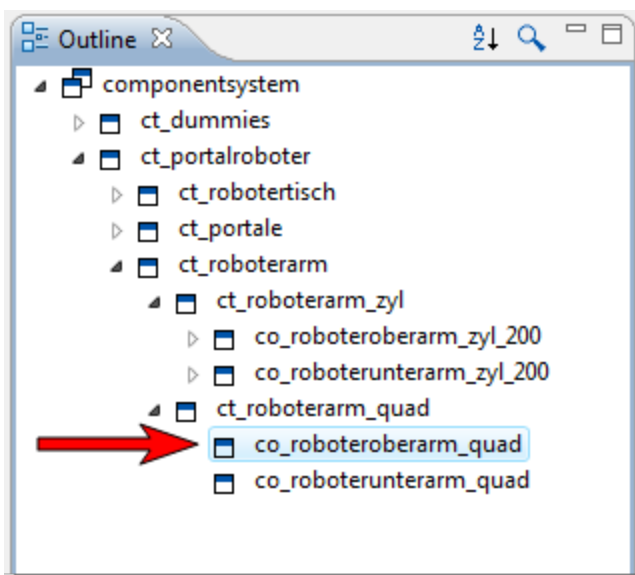
5. To create the `co_upper_robotarm_quad` and `co_lower_robotarm_quad` component types, select the `ct_robotarm_quad` component type in the **Outline** that you just created and click the **Components** button in the **Componentsystem Editor**.
6. Type `co_upper_robotarm_quad` in the **Name** field in the **New VCML Component** window and click **Finish**. The **Supertype** field should contain the `ct_robotarm_quad` entry.



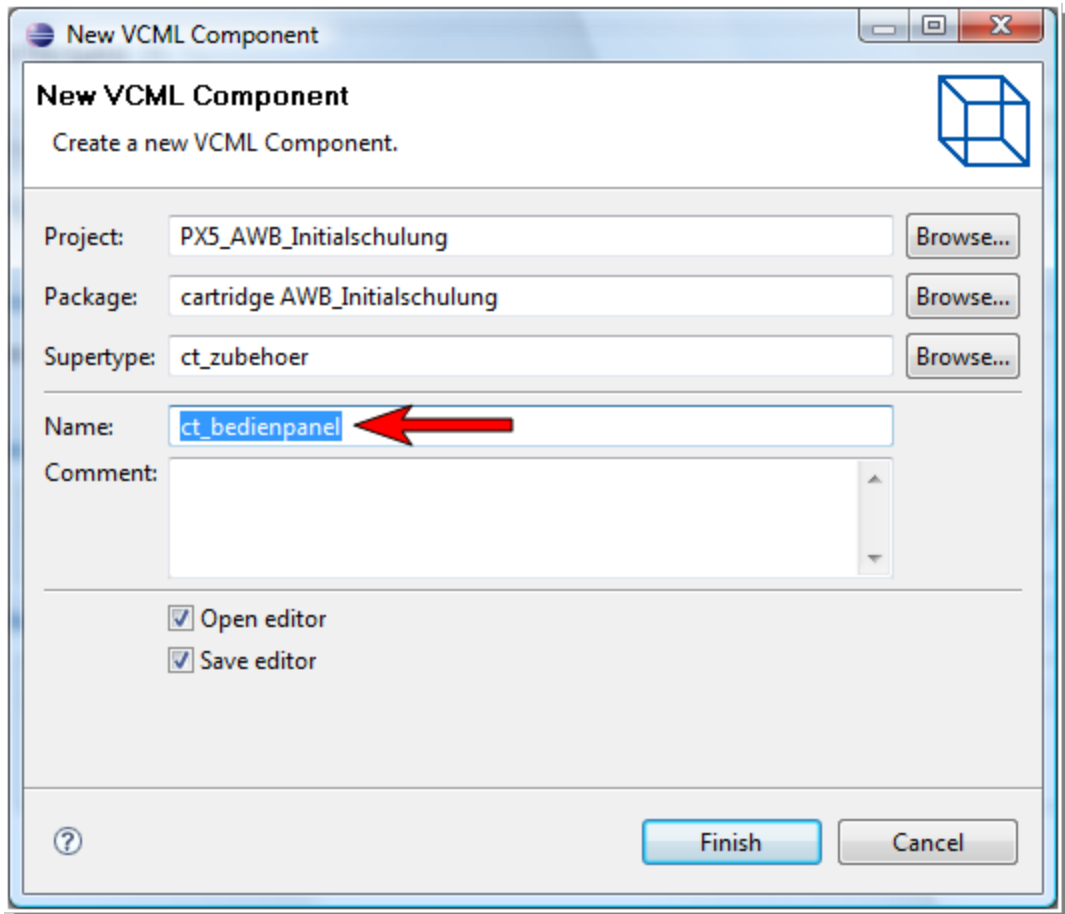
The `co_upper_robotarm_quad` component type has been created underneath the `ct_robotarm_quad` component type in the **Outline**.

Note: Note the `ct_` prefix for component type and the `co_` prefix for component.

7. Create the `co_lower_robotarm_quad` component just like you created the `co_upper_robotarm_quad` component.



8. Create the `ct_control_panel` component type underneath the `ct_supplies` component type. To do so, select the `ct_supplies` component type in the **Outline** and click the **Components** button in the **Componentsystem Editor**.
9. Type `ct_control_panel` in the **Name** field in the **New VCML Component** window and click **Finish**. The **Supertype** field should contain `ct_supplies`.

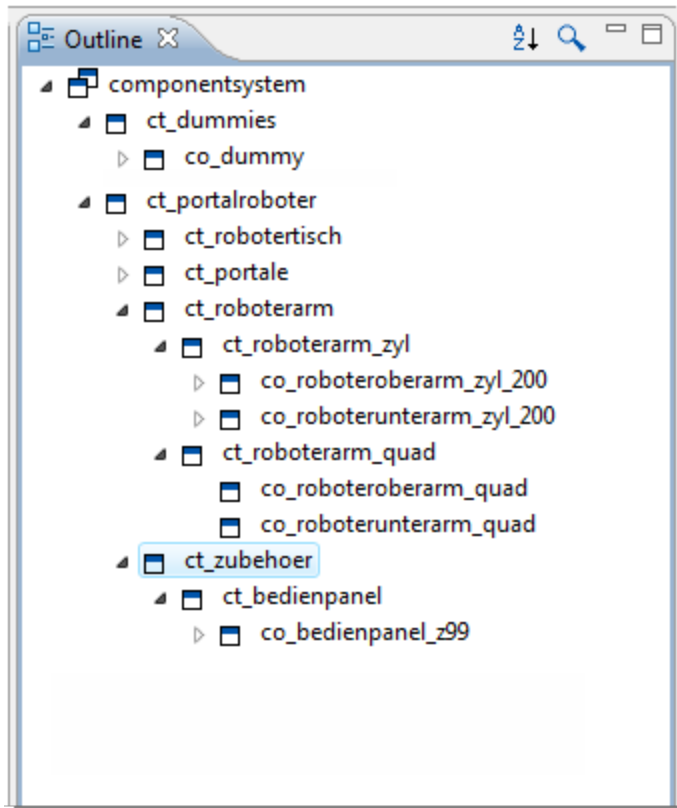


The `ct_control_panel` component type has been created underneath the `ct_supplies` component type in the **Outline**.

10. Create the `co_control_panel_99` component, just like you learned with the previous components. To do so, select the `ct_control_panel` component type in the **Outline** and click the **Components** button in the **Componentsystem Editor**.

Result

The hierarchical order of the component types and components in the **Outline** of the **Componentsystem Editor** should look like this.



Now, you can create geometries and assign them to components.

Lesson 2 - Create Geometries and Assign them to Components

In order for your newly created components to be visible in the 3D Scene as objects, you must assign geometries to them. Unlike components, geometries cannot be ordered hierarchically, which is not even necessary.

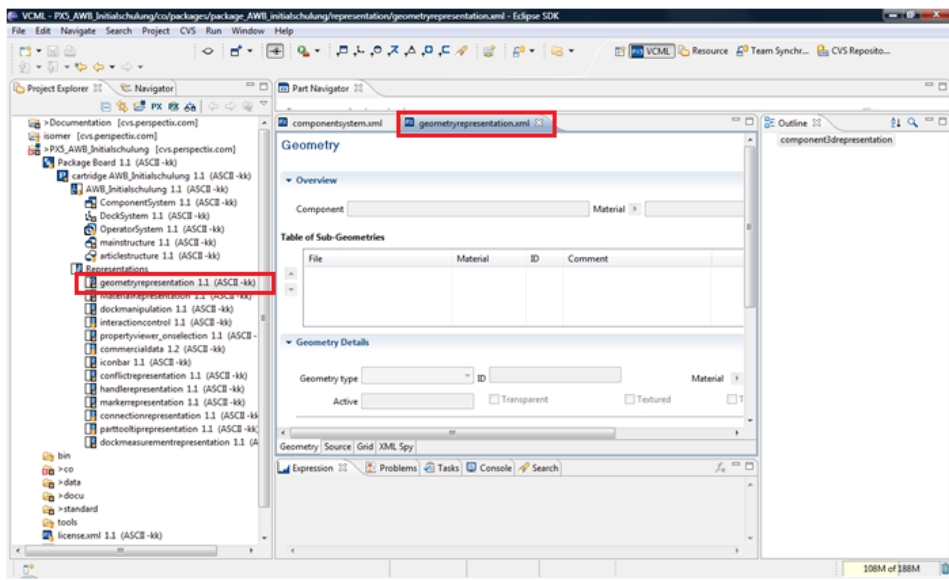
Your Assignment

Assign the following geometries to the components:

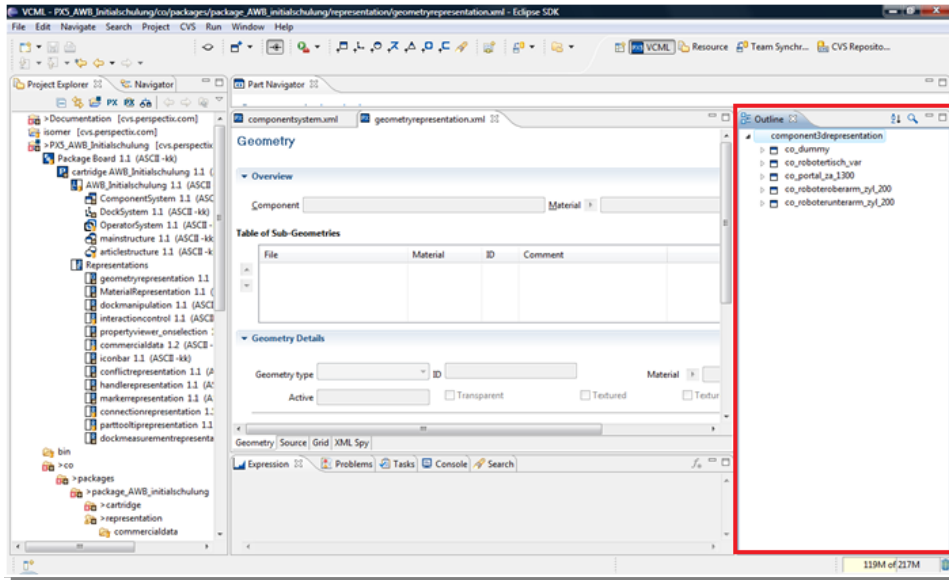
Component	Geometry File	Geometry	Position
co_upper_robotarm_quad	robot_arm_quad_upper.3d	Upper square robot arm	x_rot = 180, z_rot = 90
co_lower_robotarm_quad	robot_arm_quad_lower.3d	Lower square robot arm	
co_control_panel_99	control_panel.3d	Control panel 99	

Procedure

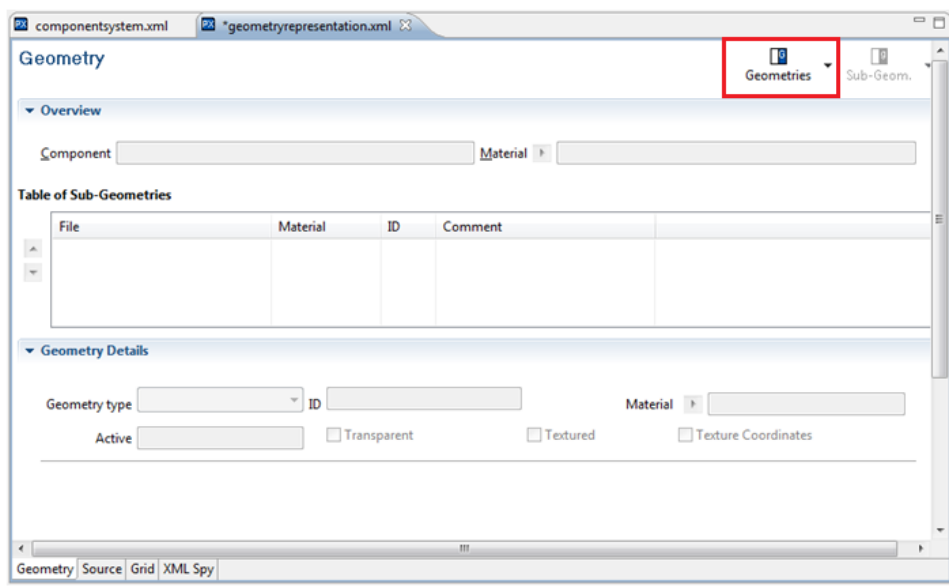
1. Open the **Geometryrepresentation Editor**.



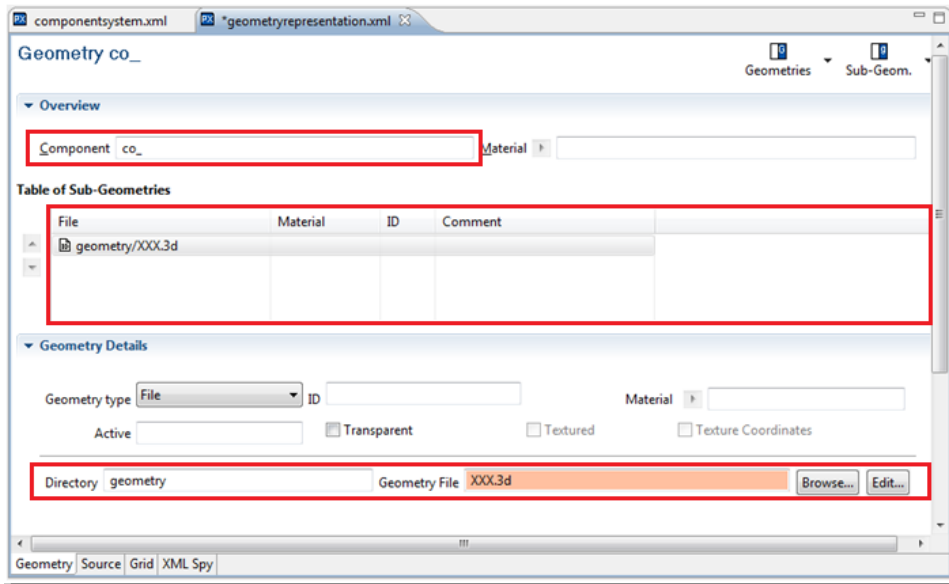
2. Take a look at the existing geometries for the robot table, the gantry and the round robot upper and lower arm in the **Outline** of the **Geometryrepresentation Editor**.



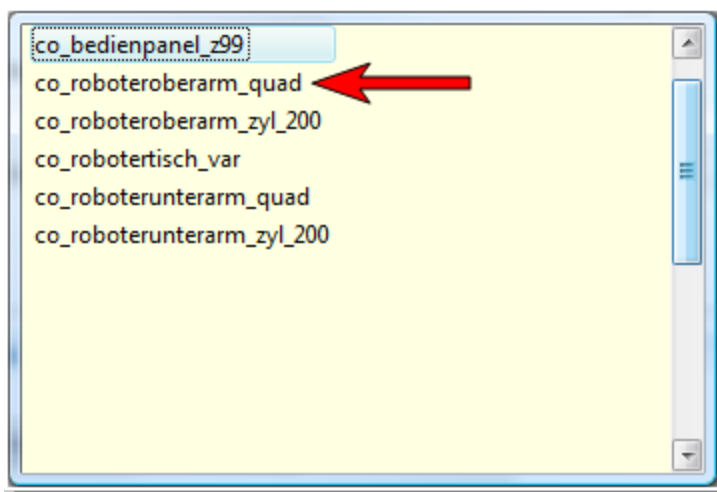
3. To create geometries for the new components, click the **Geometries** button in the **Geometryrepresentation Editor**.



4. The look of the **Geometryrepresentation Editor** changes and a few default entries appear:
 - The **Component** field contains the `co_` prefix as a hint that you should enter a component, to which the geometry should be assigned.
 - The **Table of Sub-Geometries** contains the `geometry/XXX.3d` entry. This entry refers to a `geometry` folder, but not yet to a specific file. The same entry is referenced again in the **Directory** and **Geometry File** fields.
 - The **Directory** field contains the `geometry` folder and the **Geometry File** field contains the file, which is not yet defined.



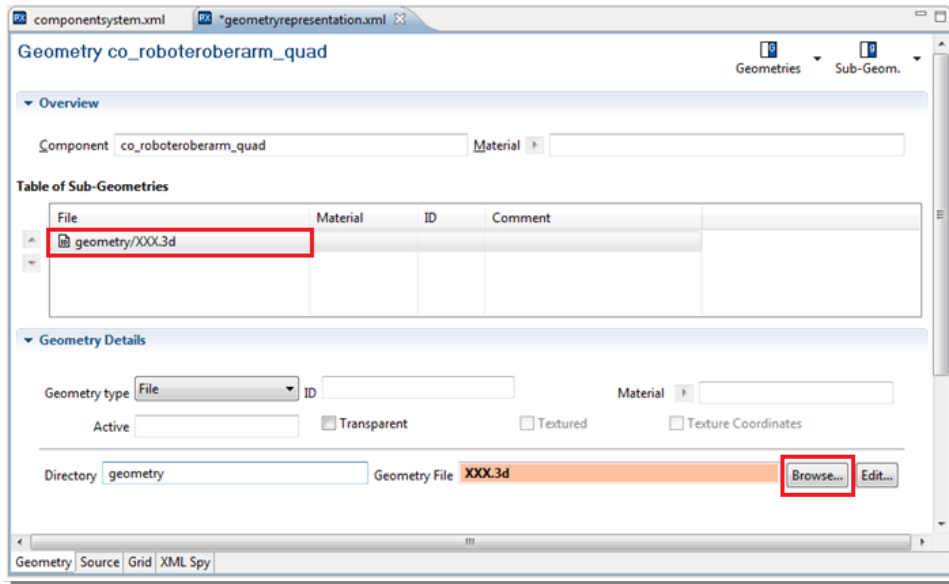
5. Type the name of the `co_upper_robotarm_quad` component in the **Component** field, place the mouse cursor behind the `co_` prefix and press CTRL+SPACEBAR, and select `co_upper_robotarm_quad` from the pop-up window.



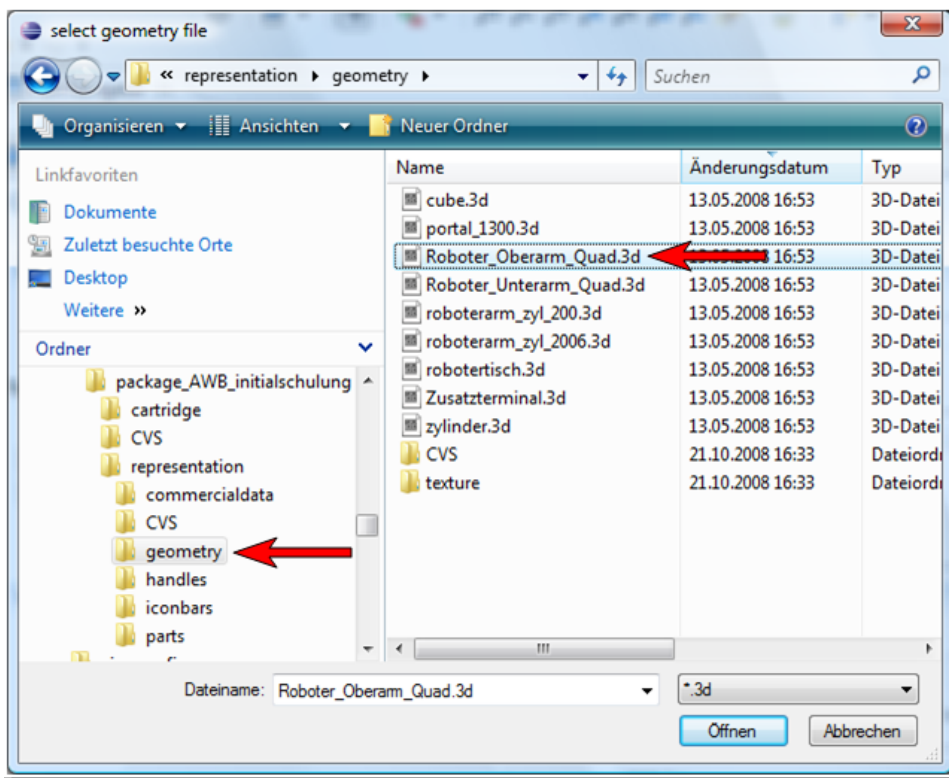
Note: The pop-up window only shows created components when the project has been rebuilt. If the components are not visible, even though they have been created, the project must be rebuilt. To do this, right-click the project and choose **Build Project** from the context menu.

6. Select the already prepared geometry in the **Table of Sub-Geometries** and click the **Browse** button.

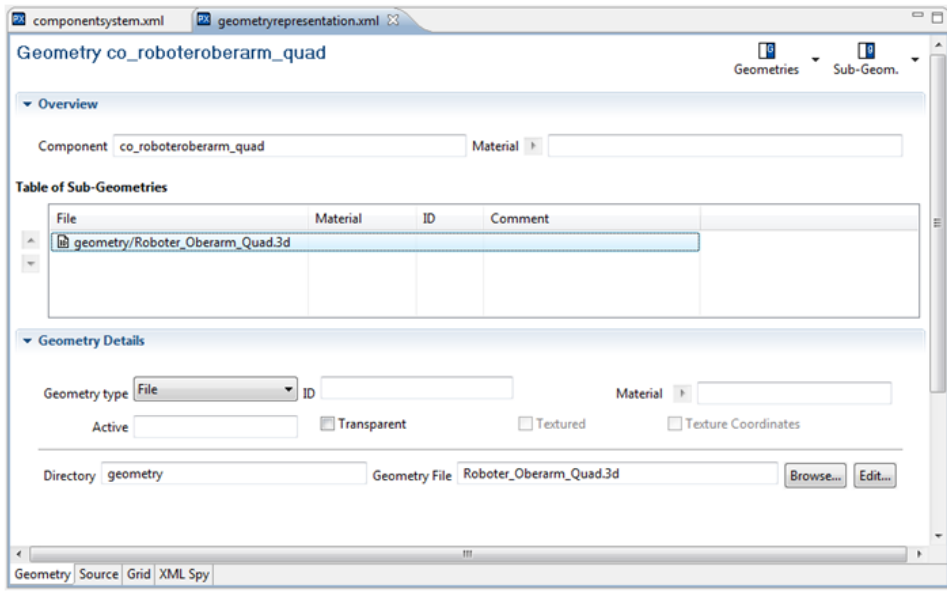
The **Explorer** with the already opened `geometry` folder opens. If the folder does not open, look for the folder and open it.



7. Select the `robot_arm_quad_upper.3d` file from the `geometry` folder in the **Explorer** and click **Open**. The `robot_arm_quad_upper.3d` file is a geometry file in which the shape of the object is saved.



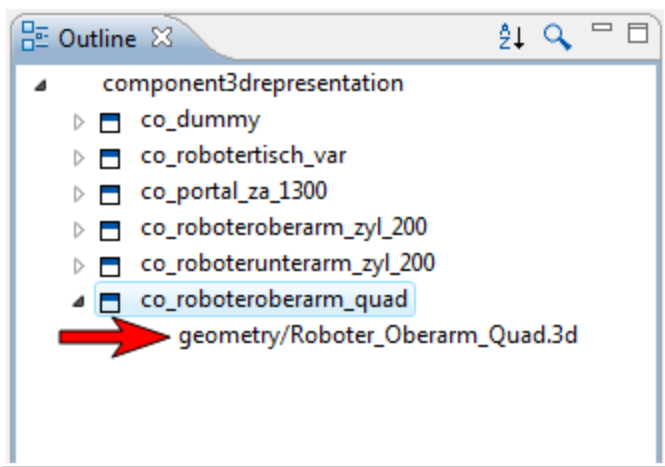
If you have done everything right, the **Geometryrepresentation Editor** should look like in the following figure.



8. Save the **Geometryrepresentation Editor**. To do this, choose **File > Save** from the main menu or click the Save icon or press CTRL+S.

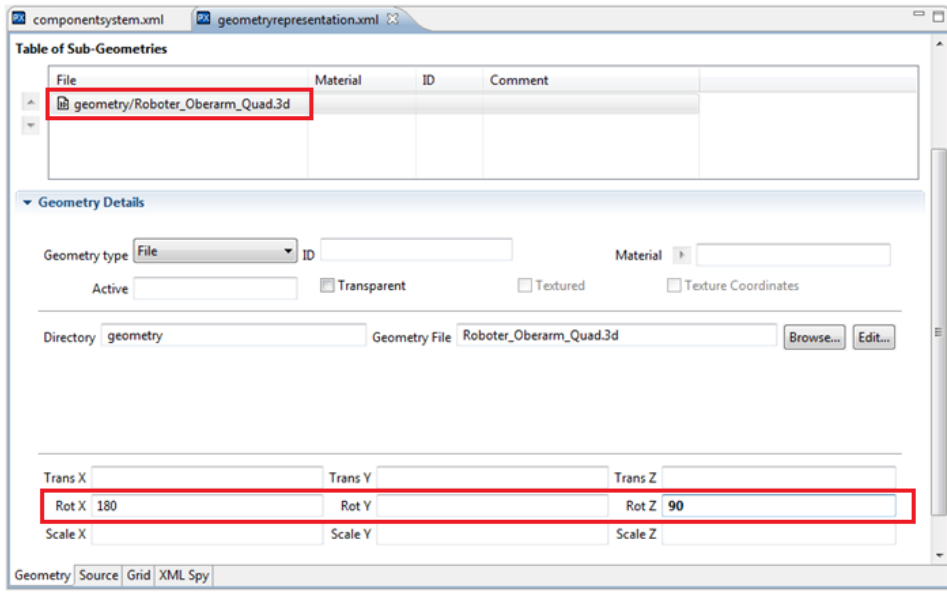
Note: Eclipse sometimes has saving problems when the mouse cursor stays in the last field that you typed something in. To solve this problem, place the mouse cursor in another field and save again. If the star next to the tab name disappears, the file has been saved.

9. Take a look at the **Outline**. The **Outline** shows that you assigned the `robot_arm_quad_upper.3d` geometry file from the `geometry` folder to the `co_upper_robotarm_quad` component.



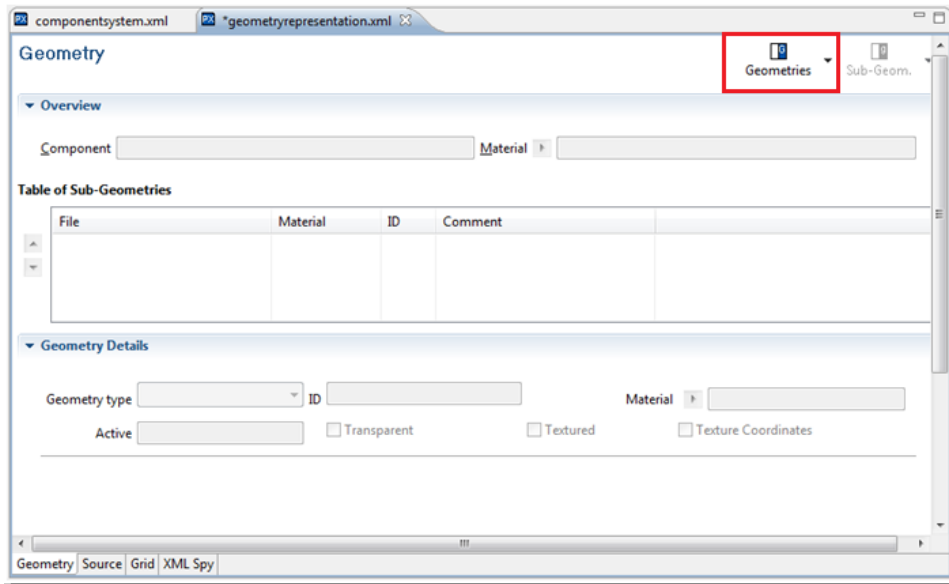
10. Now, you must position the geometry of the `co_upper_robotarm_quad` component. To do this, move the scrollbar in the **Geometryrepresentation Editor** all the way down until the fields for the positioning of the geometry appear:
 - **Trans** Translations in X-, Y- and Z-direction
 - **Rot** Rotations around the X-, Y- and Z-axis
 - **Scale** Scaling in X-, Y- and Z-direction

11. Select `robot_arm_quad_upper.3d` in the **Table of Sub-Geometries** and type the following values in the positioning fields:
 - 180 in the **Rot X** field (the geometry is rotated by 180 degrees around the X-axis)
 - 90 in the **Rot Z** field (the geometry is rotated by 90 degrees around the Z-axis)
12. Save the **Geometryrepresentation Editor**.



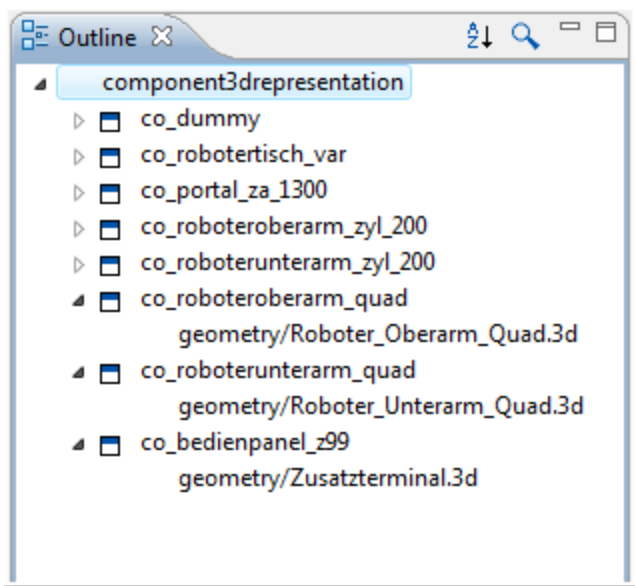
13. To assign geometries to the `co_lower_robotarm_quad` and `co_control_panel_99` components, perform the same steps that you did for the `co_upper_robotarm_quad` component:
 - a. Click the **Geometries** button in the **Geometryrepresentation Editor**.
 - b. Type the name of the component in the **Component** field and select the entry in the **Table of Sub-Geometries**.
 - c. Click the **Browse** button and open the corresponding geometry file in the **Explorer**.
 - Assign the `robot_arm_quad_lower.3d` geometry file from the `geometry` folder to the `co_upper_robotarm_quad` component.
 - Assign the `control_panel.3d` geometry file from the `geometry` folder to the `co_control_panel_99` component.

These two geometries do not need a positioning, because the origin point of the geometry has already been set optimally. Take a look at the table at the beginning of this lesson.

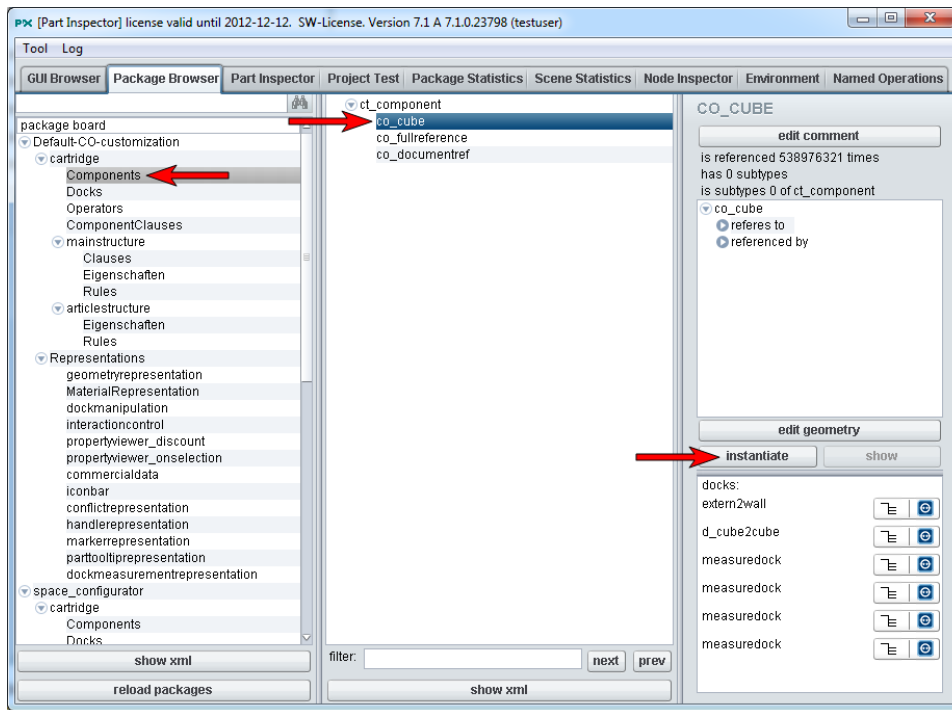


Result

1. If you have done everything right, the **Outline** of the **Geometryrepresentation Editor** should contain the following entries.



2. To see your newly created geometries, start the **Configurator** and press SHIFT+F12.
The **Inspector** with all components and their assigned files opens.
3. Select **Components** in the left pane in the **Inspector**.
The middle pane shows the same structure as the **Outline**.
4. Select a newly created component in the middle pane, i.e. an entry with the `co_` prefix (you did not assign a geometry to the others) and click the **instantiate** button to the right.



The created geometry appears in the **3D Scene** of the **Configurator**.

Note: If the geometry does not appear in the 3D Scene, press the SPACEBAR to focus.

Now, you can create docks.

Lesson 3 - Create Docks

In order to connect the geometric objects in the 3D Scene, you must create docks and assign them to components.

Docks are snapping points where the geometric objects can be connected. For every dock you must define a partnerdock. The dock is attached to a component and the partnerdock is attached to the component with which it is supposed to connect.

Remember that the geometric objects represent components. When you drag an object onto another object in the 3D Scene, the corresponding components connect at the points where the docks were placed.

Your Assignment

Create the following docks:

Dock	Degree of Freedom	Assigned Component
do_controlpanel_2_gantry		ct_control_panel
do_gantry_2_controlpanel	DOF type 'rotation DOF' / DOF axis 'z' / From '0' to '360'	co_gantry_belt_drive
do_arm_quad_2_gantry		co_upper_robotarm_quad
do_gantry_2_arm_quad	DOF type 'translation DOF' / DOF axis 'y' / From '-400' to '400'	co_gantry_belt_drive
do_lower_armquad_2_upper_armquad		co_lower_robotarm_quad
do_upper_armquad_2_lower_armquad	DOF type 'rotation DOF' / DOF axis 'x' / From '-45' to '45'	co_upper_robotarm_quad

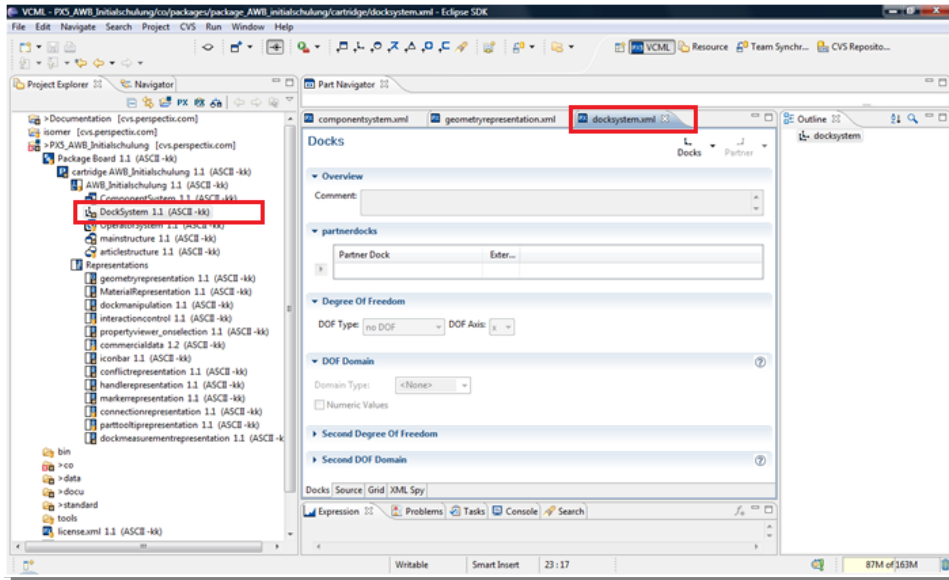
Docks have specific names. Each dock is attached to a component and can be connected to another component using a partnerdock. The name of a dock consists of three terms, e.g. `do_controlpanel_2_gantry`. Docks contain the `do_` prefix, so that they can easily be distinguished from other elements in the XML code.

- The first term refers to the component, to which the dock is attached to, e.g. the control panel.
- The middle term consists of an underscore, the number 2, and another underscore (`_2_`).
- The third term refers to the component, to which the partnerdock is attached to, e.g. the gantry.

Therefore, the name of the dock `do_controlpanel_2_gantry` signifies that the control panel can be connected to the gantry. You can always recognize a dock in the code on the basis of its `do_` prefix and its middle term `_2_`.

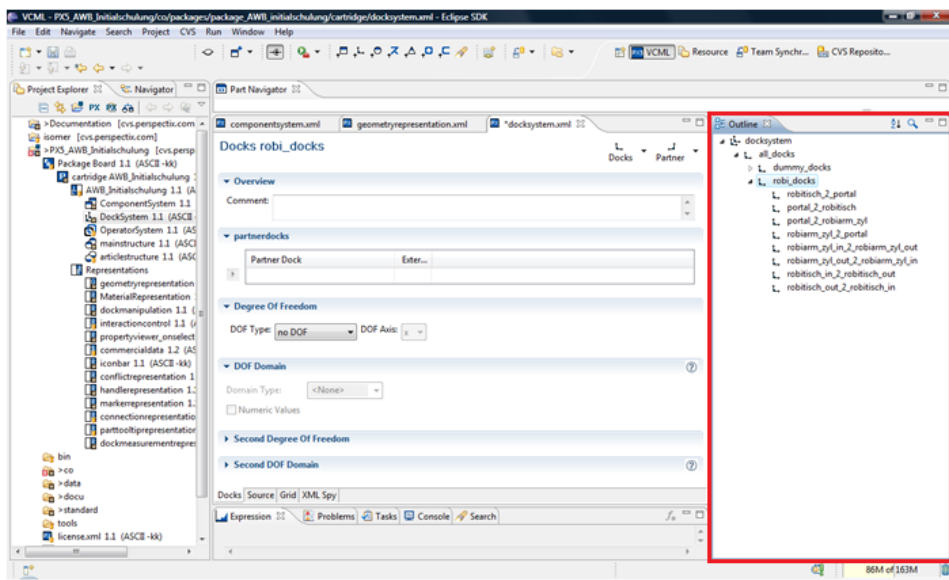
Procedure

1. Open the **Docksystem Editor**.



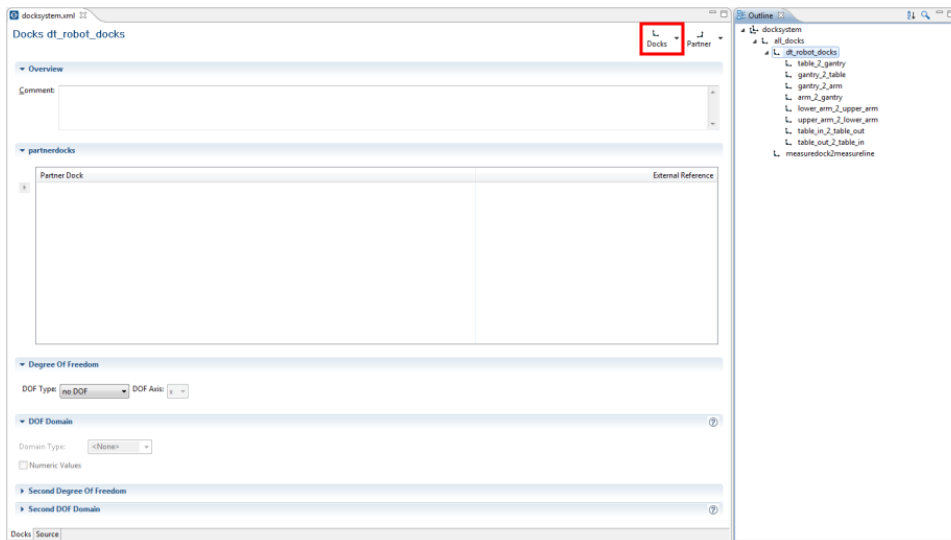
2. Take a look at the existing and assigned docks in the **Outline** of the **DocksysteM Editor**.

Tip: If you want to try out how docks work, open the **Configurator**, drag the gantry and the robot table from the **Iconbar** into the **3D Scene**, and then drag the gantry onto the robot table. The gantry and the robot table will connect. Just like these docks, you will also attach docks to your new components.



3. Select the `dt_robot_docks` supertype in the **Outline** and click the **Docks** button in the **DocksysteM Editor**.

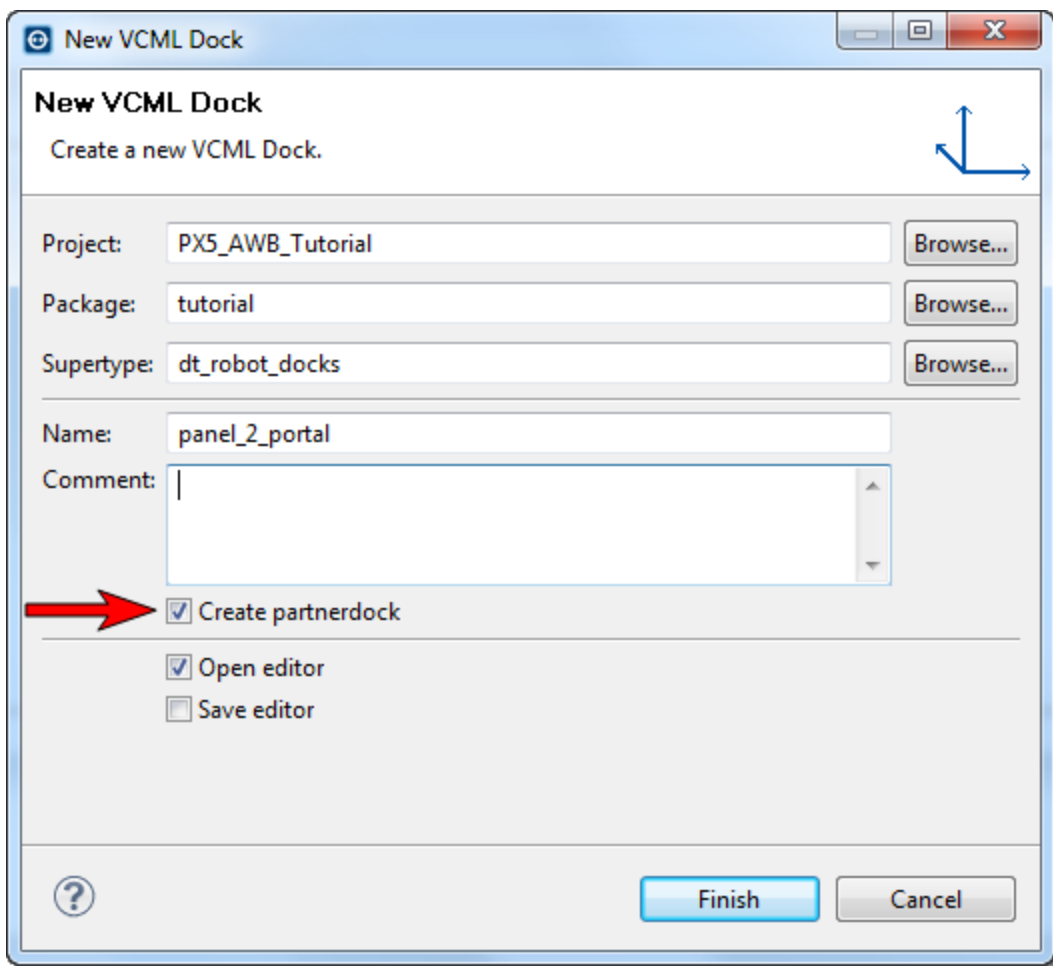
Note: The supertype contains the `dt_` prefix, i.e. `dt_robot_docks` is a supertype and all docks are hierarchically below it. Docks, like components, can be hierarchically ordered and grouped into supertypes and types.



4. Type `do_gantry_2_controlpanel` in the **Name** field in the **New VCML Docks** window for the first dock to be created and click the **Finish** button. The **Project**, **Package** and **Supertype** fields already contain entries.

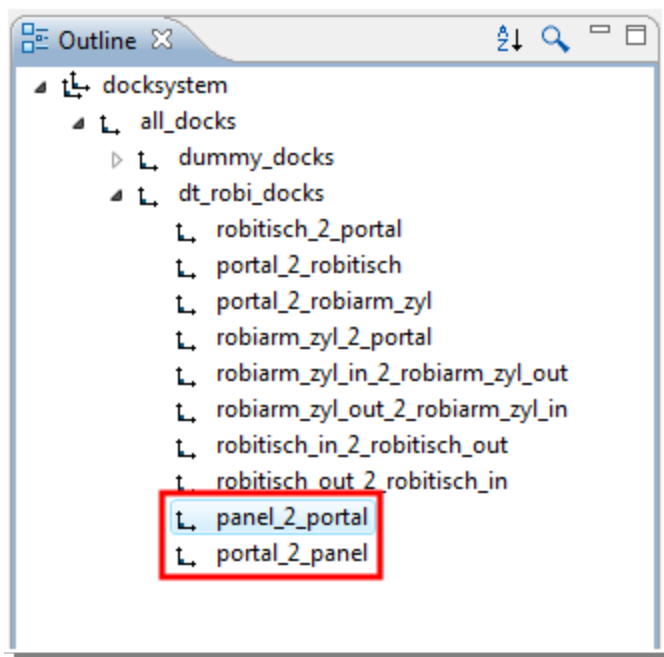
If you want to immediately create a partnerdock for your dock, select the **Create Partnerdock** checkbox. In order to create the partnerdock, the name of the first dock must look like this:

`[dockname1]2[dockname1]` or `[dockname1]_2_[dockname1]` or `[dockname1]_to_[dockname1]`

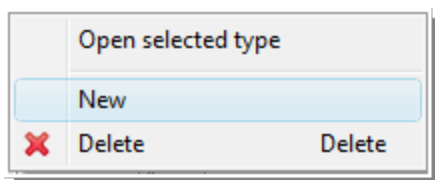


5. If you did not create a partnerdock in the previous step, create the `do_controlpanel_2_gantry` partnerdock for the `do_gantry_2_controlpanel` dock:
 - a. Select the `dt_robot_docks` supertype in the **Outline** and click the **Docks** button in the **Docksystem Editor**.
 - b. Type `do_controlpanel_2_gantry` in the **Name** field in the **New VCML Dock** window for the partnerdock and click **Finish**.

If you have done everything right, the **Outline** should contain two new entries: The `do_gantry_2_controlpanel` dock and its partnerdock `do_controlpanel_2_gantry`.



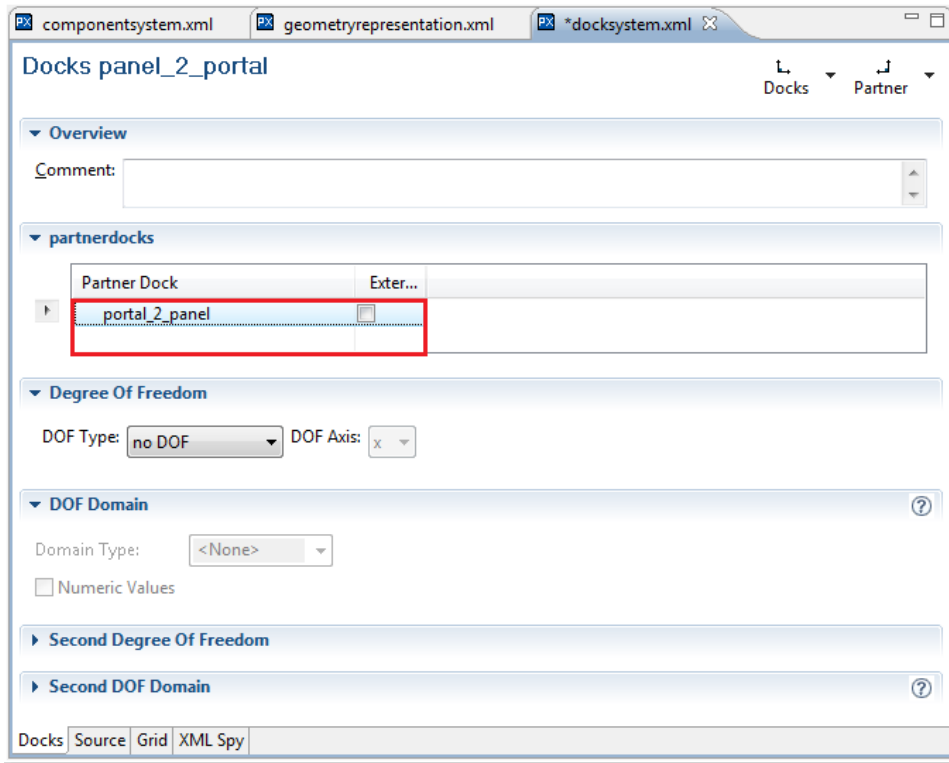
6. Select the newly created `do_controlpanel_2_gantry` dock in the **Outline**.
7. Assign the partnerdock to this dock. To do this, right-click inside the **partnerdocks** table in the **Docksystem Editor** and choose **New** from the context menu.



Note: If the table is closed, open it using the arrow to the left of the entry.

A row in the **partnerdocks** table opens.

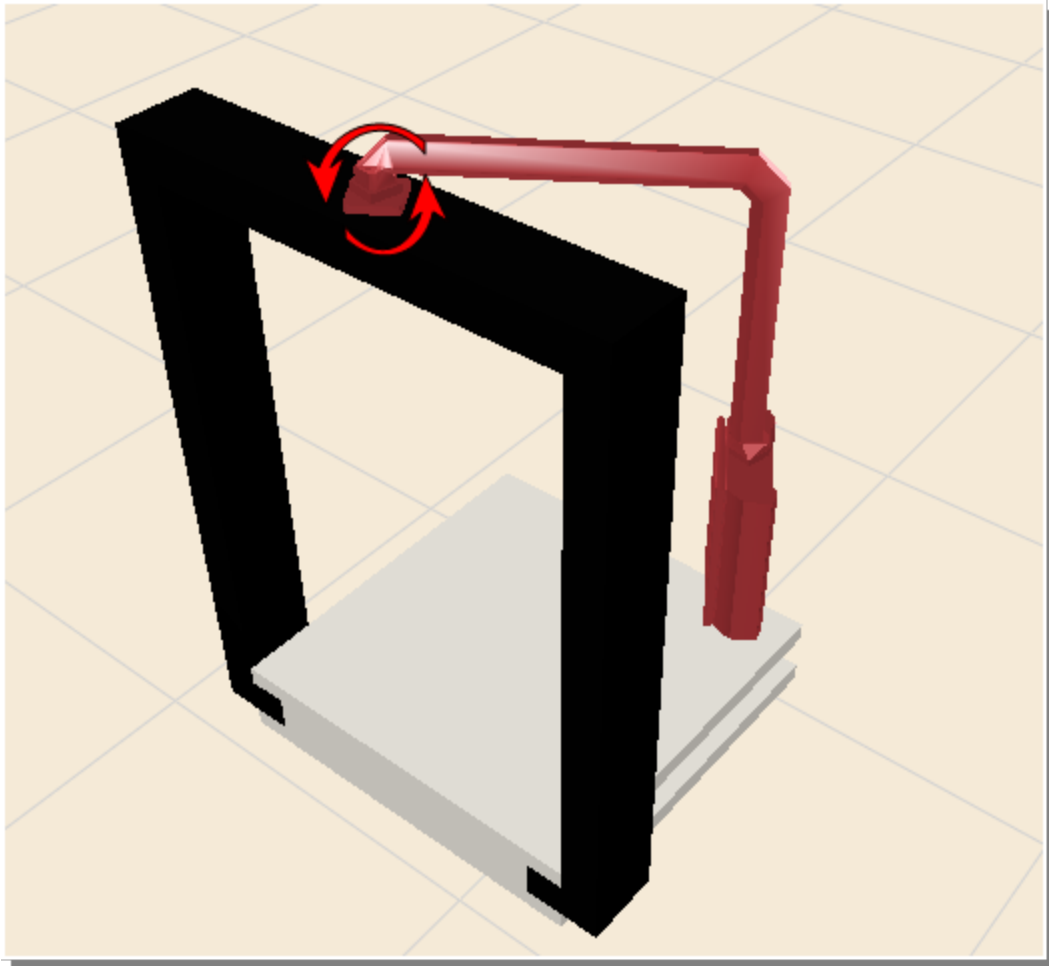
8. Type the name of the `do_gantry_2_controlpanel` partnerdock in the row or press CTRL+SPACEBAR and select the partnerdock from the pop-up window.
9. Save the **Docksystem Editor**.



10. If you select the `do_gantry_2_controlpanel` dock in the **Outline** now, you will see the `do_controlpanel_2_gantry` entry in the **partnerdocks** table in the **Docks system Editor**. The partnerdock has automatically been ported. The two docks are now partners and can be attached to two different components. If you drag the geometry of one of these components onto the geometry of the other component in the **3D Scene**, their docks will connect.

However, we are not that far yet. First, you must attach the `do_gantry_2_controlpanel` dock to the gantry and the `do_controlpanel_2_gantry` dock to the control panel (`co_control_panel_99`).

As you can see in the following figure, the control panel should turn around the gantry. You must define this first at the corresponding dock, i.e. you must assign degrees of freedom (DOFs) to the dock. We call these docks DOF docks.

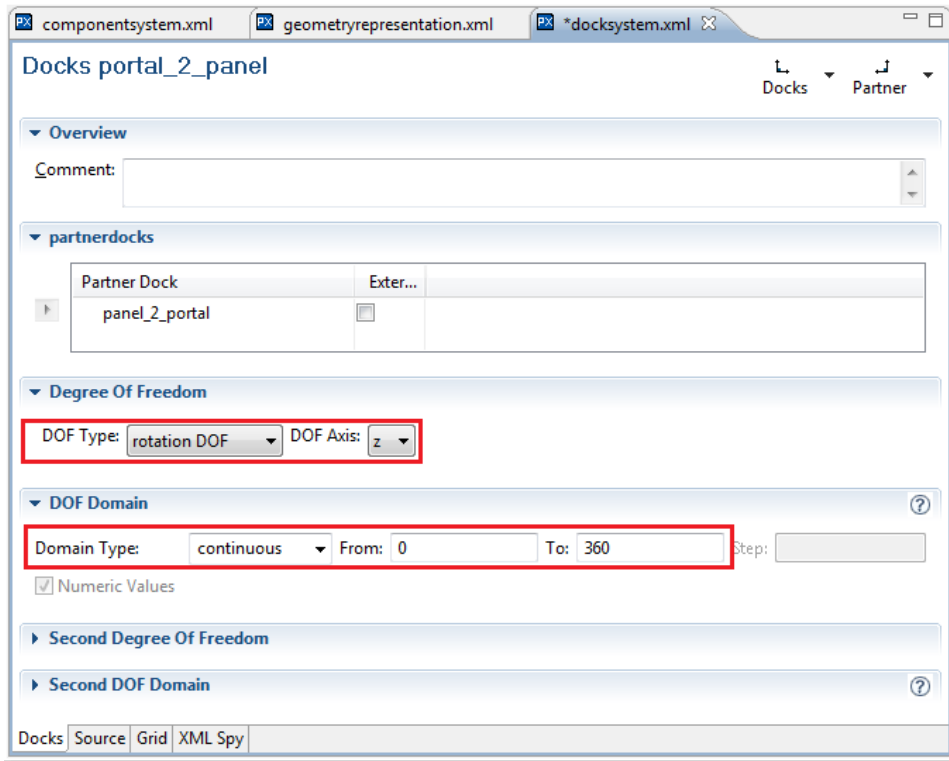


11. To assign degrees of freedom to the `do_gantry_2_controlpanel` dock, select `rotation` DOF in the **DOF Type** list box and `z` for the z-axis in the **DOF Axis** list box.

This means that your `do_gantry_2_controlpanel` dock can rotate around the z-axis.

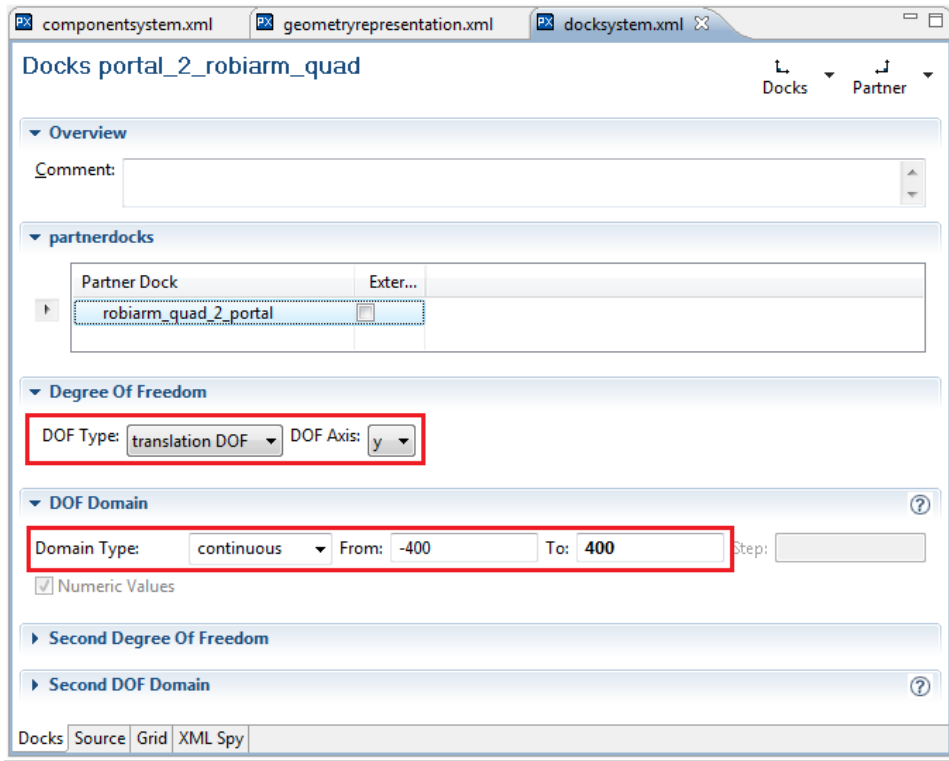
12. Select `continuous` in the **Domain Type** list box and type `0` for the angular degree in the **From** field and `360` for the angular degree in the **To** field.

This means that your `do_gantry_2_controlpanel` dock may continuously turn from 0 to 360 degrees.



Note: Don't let the fact that `do_gantry_2_controlpanel` is written in the **partnerdocks** table confuse you. In the upper left corner of the **Docksystem Editor** you can see the correct name of the dock that you are editing.

13. Now, create the other docks in the same way:
 - `do_arm_2_gantry`
 - `do_gantry_2_arm_quad`
 - `do_lower_armquad_2_upper_armquad`
 - `do_upper_armquad_2_lower_armquad`
14. Save the **Docksystem Editor**.
15. Define degrees of freedom for the `do_gantry_2_arm_quad` dock. To do this, select `translation DOF` in the **DOF Type** list box and `y` in the **DOF Axis** list box.
This means that a movement in the y-direction is possible for this dock.
16. Select `continuous` in the **Domain Type** list box and type `-400` in the **From** field and `400` in the **To** field.
This means that the movement on the y-axis is limited from `-400` to `400`.
17. Save the **Docksystem Editor**.



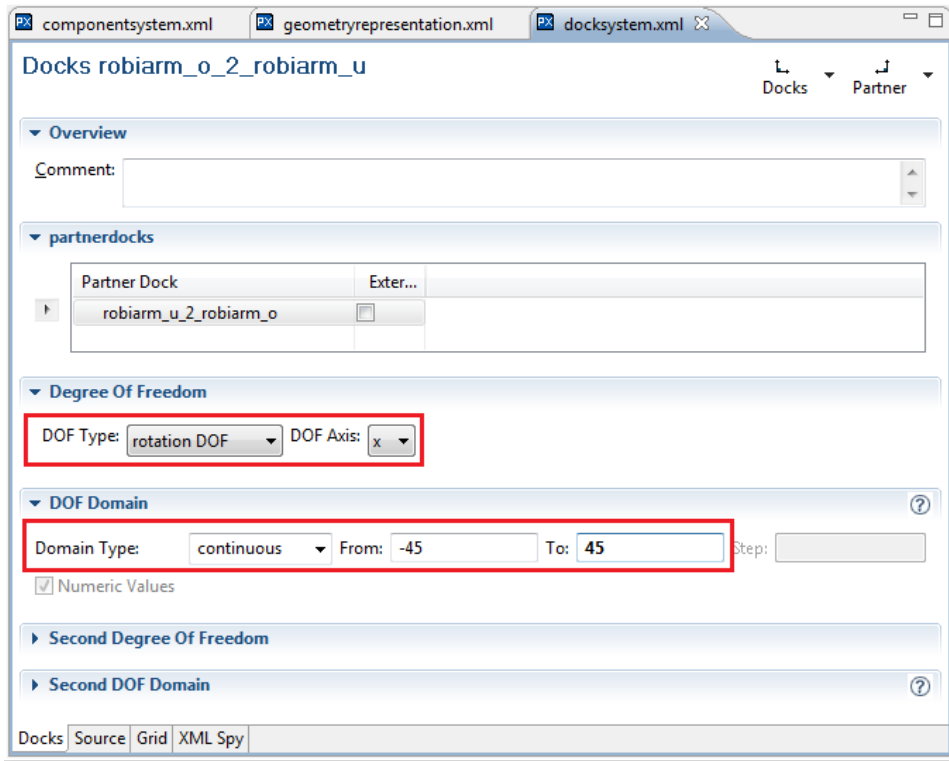
18. Define degrees of freedom for the `do_upper_armquad_2_lower_armquad` dock also. To do this, select `rotation DOF` in the **DOF Type** list box and `x` in the **DOF Axis** list box.

This means that a movement in the x-direction is possible for this dock.

19. Select `continuous` in the **Domain Type** list box and write `-45` in the **From** field and `45` in the **To** field.

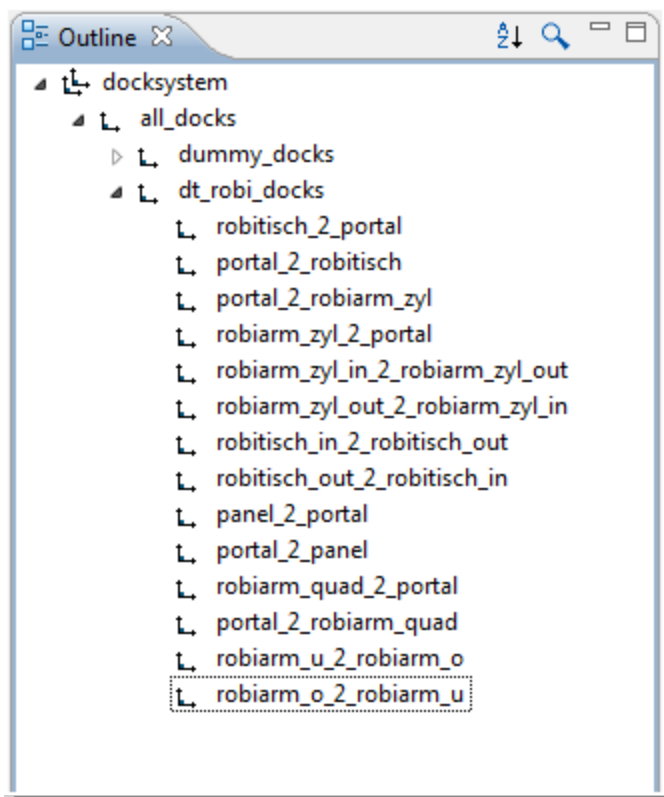
This means that the movement on the x-axis is limited from `-45` to `45`.

20. Save the **Docksystem Editor**.



Result

If you have done everything right, your Outline should look like in the following figure. The order of the docks is not important. What's important is that they exist and that they are ordered under the correct supertype.



Note: When you create a new dock, first select the `dt_robot_docks` supertype in the Outline. The new docks will be created hierarchically under this dock.

The necessary docks are now ready and must be assigned to the components, so that the components can be assembled. In the next lesson you will learn how to assign your newly created docks to the components.

Now, you can assign docks to components.

Lesson 4 - Assign Docks to Components

In the previous lesson you created docks. Now you must assign the docks to the components. This means that the components will receive coordinates, where the individual docks are attached to. Then the geometric objects can be connected (docked) with each other in the 3D Scene.

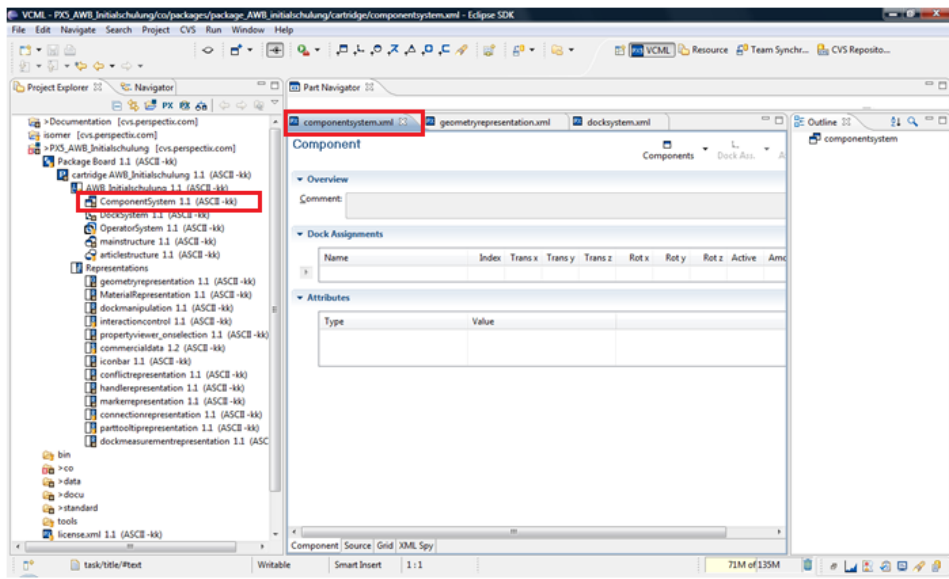
Your Assignment

Assign the following docks to the components:

Component	Dock	Trans x	Trans y	Trans z	Rot x	Rot y	Rot z
co_control_panel_99	do_controlpanel_2_gantry	0	0	0	0	0	0
co_gantry_belt_drive	do_gantry_2_controlpanel	100	0	1600	0	0	0
co_lower_robotarm_quad	do_lower_armquad_2_upper_arm-quad	0	0	-100	180	0	0
co_upper_robotarm_quad	do_upper_armquad_2_lower_arm-quad	0	0	-300	0	0	-90
co_upper_robotarm_quad	do_arm_quad_2_gantry	0	0	0	0	0	0
co_gantry_belt_drive	do_gantry_2_arm_quad	100	0	1500	0	0	0

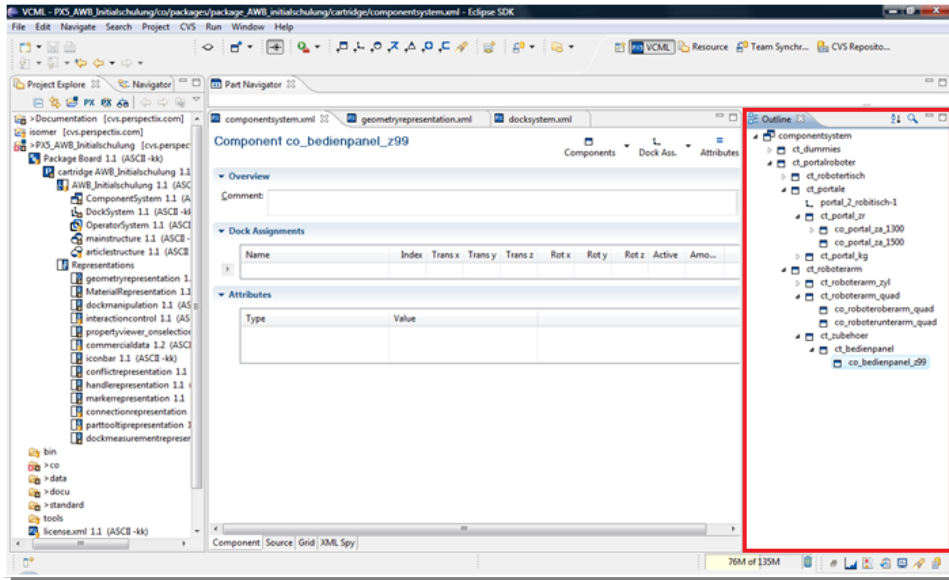
Procedure

1. Open the **Componentsystem Editor**. If the **Componentsystem Editor** is already open, click the **componentsystem.xml** tab.

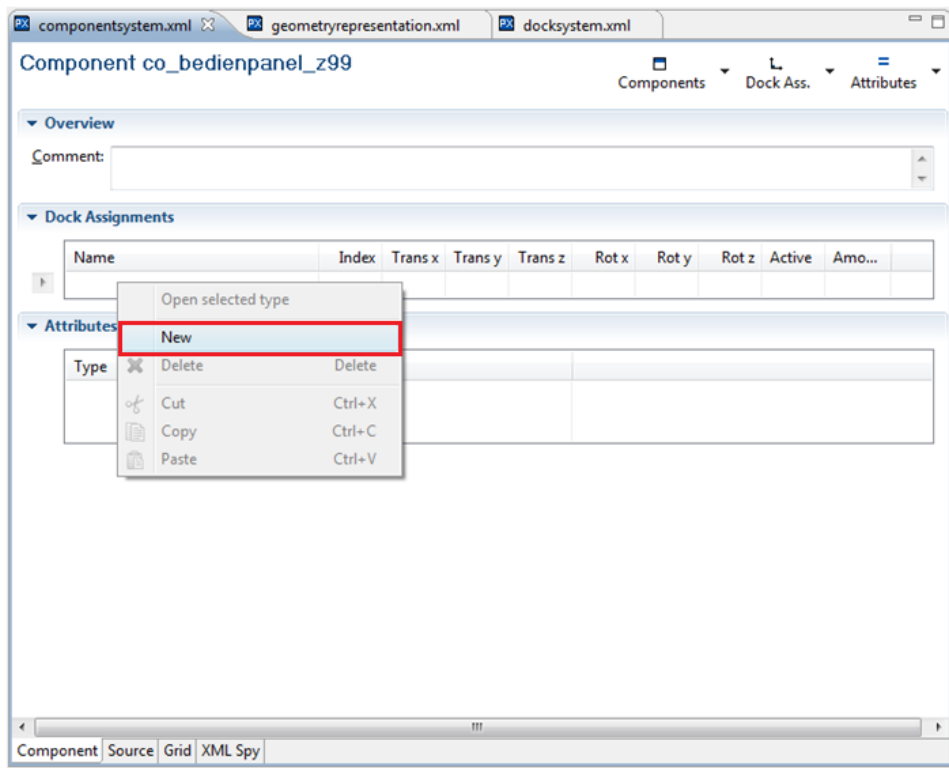


2. Look for the following components in the **Outline** of the **Componentsystem Editor**:

- co_control_panel_99
- co_upper_robotarm_quad
- co_lower_robotarm_quad
- co_gantry_belt_drive

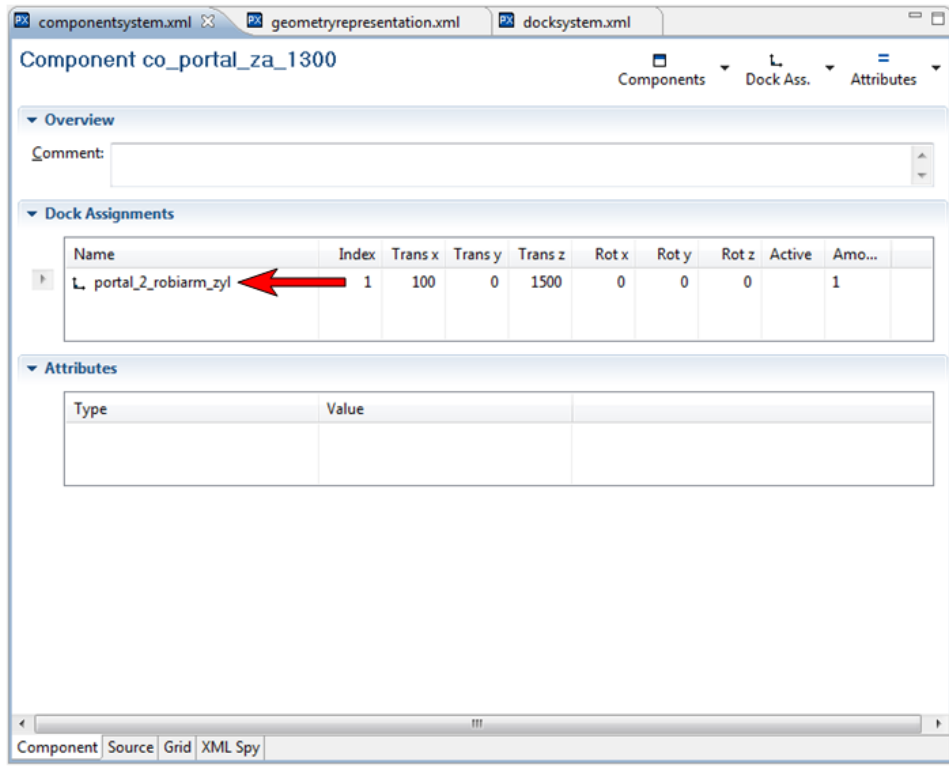


- First, you must create a connection between the control panel and the gantry. This means that you must assign the `do_controlpanel_2_gantry` dock to the `co_control_panel_99` component and the `do_gantry_2_controlpanel` dock to the `co_gantry_belt_drive` component. To do this, select the `co_control_panel_99` component in the **Outline** of the **Componentsystem Editor**.
- Right-click inside the **Dock Assignments** table in the **Component System Editor** and choose **New** from the context menu.
A row opens in the **Dock Assignments** table.
- Type `do_controlpanel_2_gantry` and the coordinates `0 / 0 / 0` in the row. This means that the dock lies at the origin point of the component.



6. Save the **Componentsystem Editor**.
7. Now, you must attach the `do_gantry_2_controlpanel` partnerdock to the gantry for the `do_controlpanel_2_gantry` dock, which is attached to the control panel. Then these two geometric objects can be connected at these points in the **3D Scene**. To do this, select the `co_gantry_belt_drive` component in the **Outline** of the **Componentsystem Editor**.

The **Dock Assignments** table already contains the `do_gantry_2_arm_cyl` dock. You must add another dock to this existing dock. The gantry can then be connected to geometric objects at more than one place.



8. Right-click inside the **Dock Assignments** table in the **Component System** and choose **New** from the context menu.

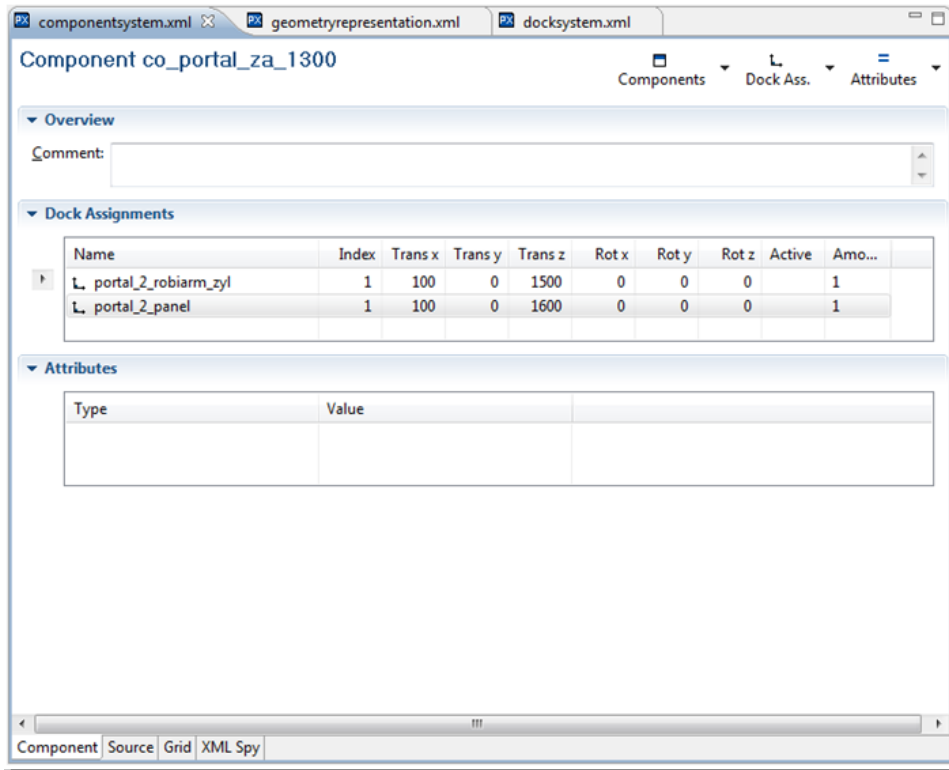
A row opens in the **Dock Assignments** table.

9. Type `do_gantry_2_controlpanel` in the new row of the **Name** column, 100 in the **Trans x** column and 1600 in the **Trans z** column.

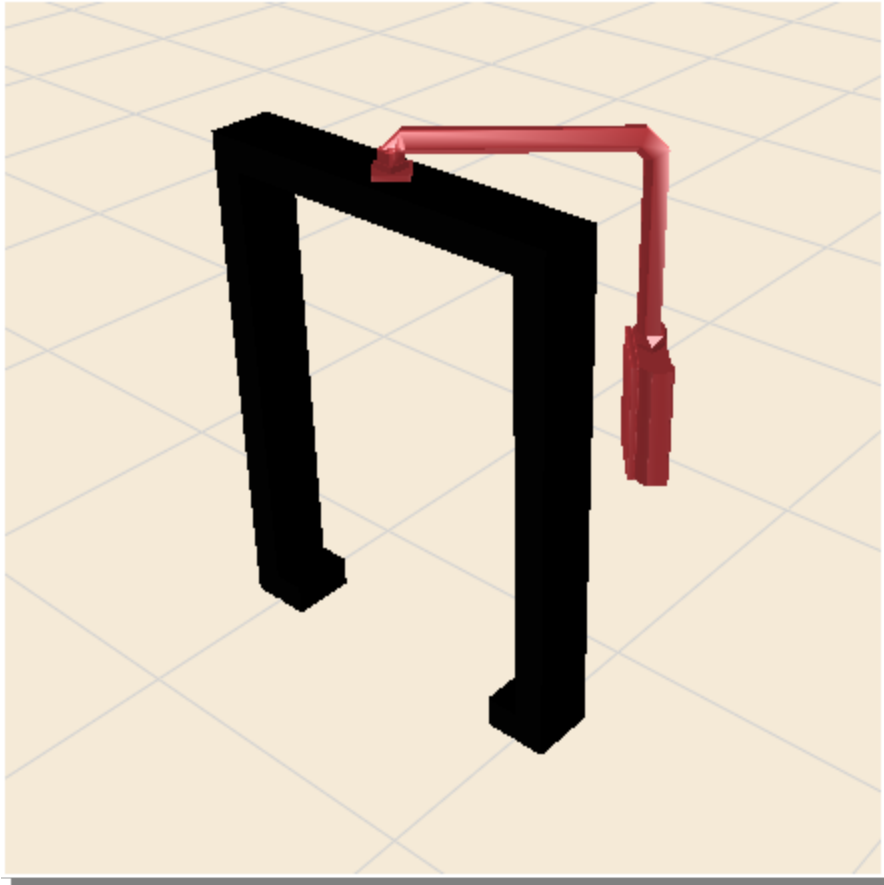
This means that the dock, the place where other components can be connected, lies 100 units in the X-direction and 1600 units in the Y-direction of the origin point of the component.

10. Save the **Componentsystem Editor**.

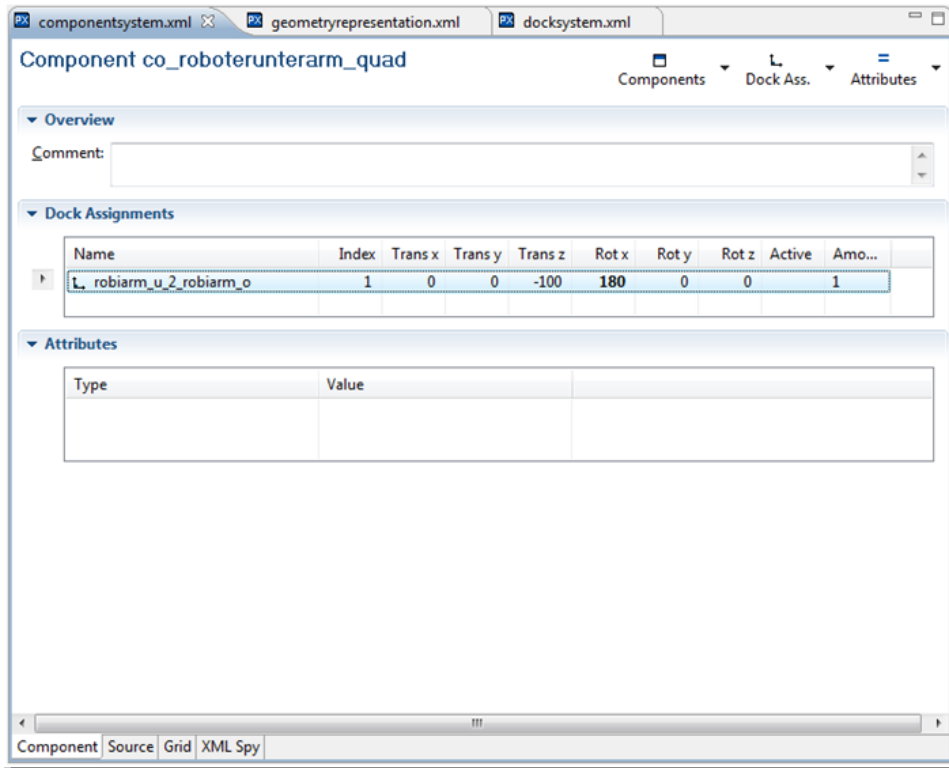
If you have done everything right, your **Componentsystem Editor** should look like in the following figure.



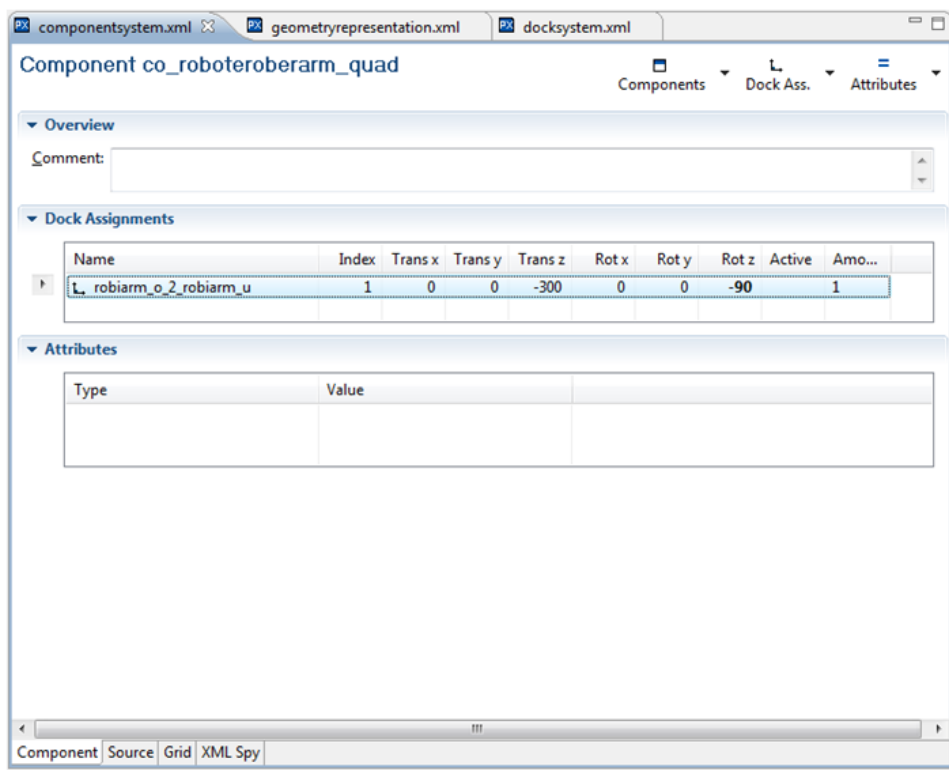
11. Press F5 to update the **Configurator**.
12. Connect the co_gantry_belt_drive component, the gantry and the co_control_panel_99 control panel in the **Configurator**.



13. Now assign the docks to the `co_upper_robotarm_quad` and `co_lower_robotarm_quad` components. To do this, select the `co_lower_robotarm_quad` component in the **Outline** of the **Componentsystem Editor**.
14. Right-click inside the **Dock Assignments** table in the **Component System** and choose **New** from the context menu.
A row opens in the **Dock Assignments** table.
15. Type `do_lower_armquad_2_upper_armquad` in the **Name** column, `-100` (units) in the **Trans z** column and `180` (degrees) in the **Rot x** column.
16. Save the **Componentsystem Editor**.



17. Do the same thing you did for the co_lower_robotarm_quad robot lower arm for the co_upper_robotarm_quad robot upper arm and assign the do_upper_armquad_2_lower_armquad dock with the **Trans z** -300 and **Rot z** -90 coordinates.
18. Save the **Componentsystem Editor**.



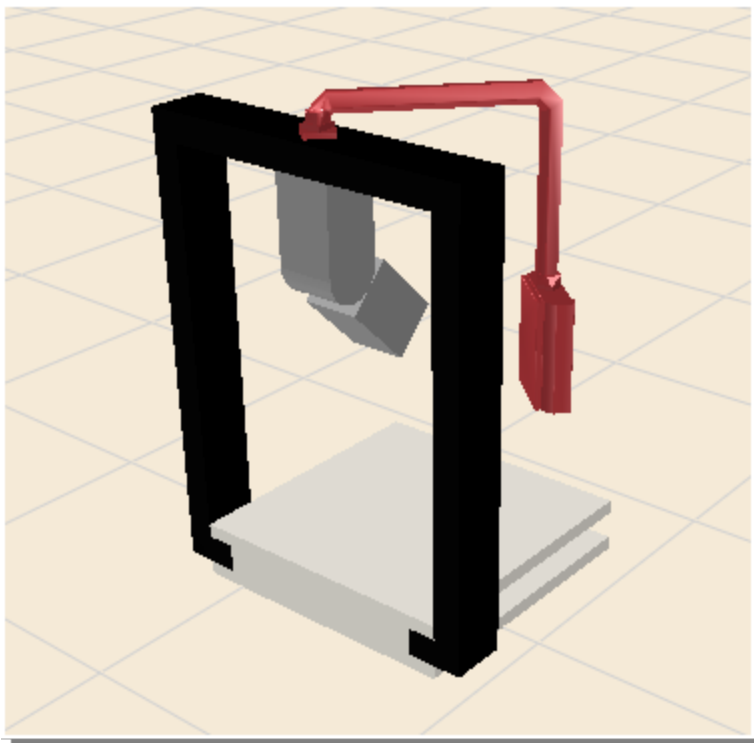
19. To also be able to connect the upper robot arm to the gantry, you must attach another dock to the upper robot arm and the corresponding partnerdock to the gantry. To do this, select the `co_upper_robotarm_quad` component again in the **Outline** of the **Componentsystem Editor** and type `do_arm_quad_2_gantry` in the **Dock Assignments** table in the **Component System Editor**.

This dock does not need any other coordinates.

20. Save the **Componentsystem Editor**.
21. To be able to attach the upper robot arm to the gantry, you must attach the partnerdock to the gantry. To do this, select the `co_gantry_belt_drive` component in the **Outline** of the **Componentsystem Editor** and type `do_gantry_2_arm_quad` with the coordinates **Trans x** 100 and **Trans z** 1500 (units) in the **Dock Assignments** table in the **Componentsystem Editor**.
22. Save the **Componentsystem Editor**.

Result

Now you can connect all parts.



To be able to drag the geometric objects from the Iconbar into the 3D Scene, you must create icons and place the components in the Iconbar. In the next lesson, you will learn how to create icons and how to assign them to components.

Now, you can create icons and assign them to components.

Lesson 5 - Create Icons and Assign them to Components

First, you created components for the control panel and the robot arms. Then, you assigned geometries and docks to the components. To be able to drag the geometric objects in the Configurator from the Iconbar into the 3D Scene, you must create icons and assign them to components.

Your Assignment

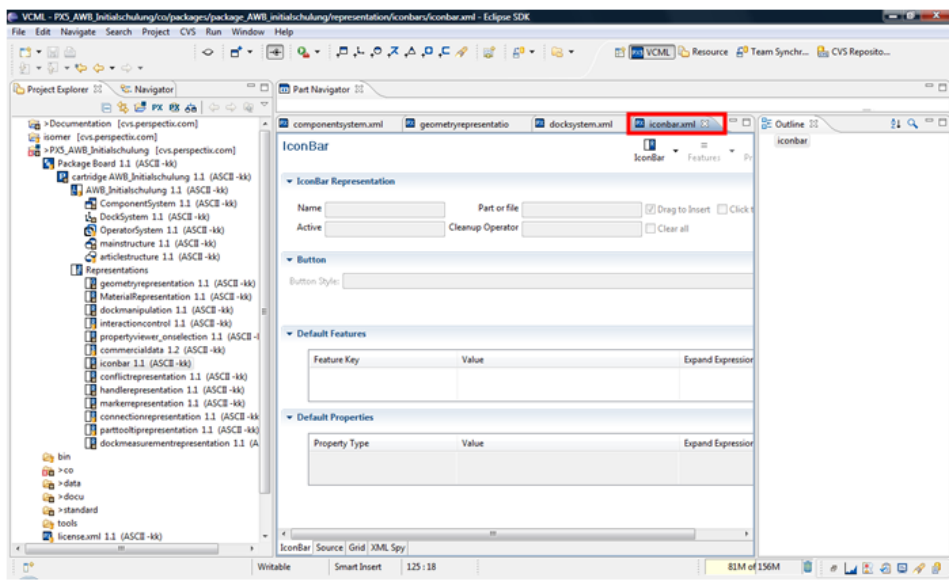
Create the following icons and assign them to the components:

Icon	Part or File
Control Panel Z99	co_control_panel_99
Square Robot Arm	robotarm_quad.par

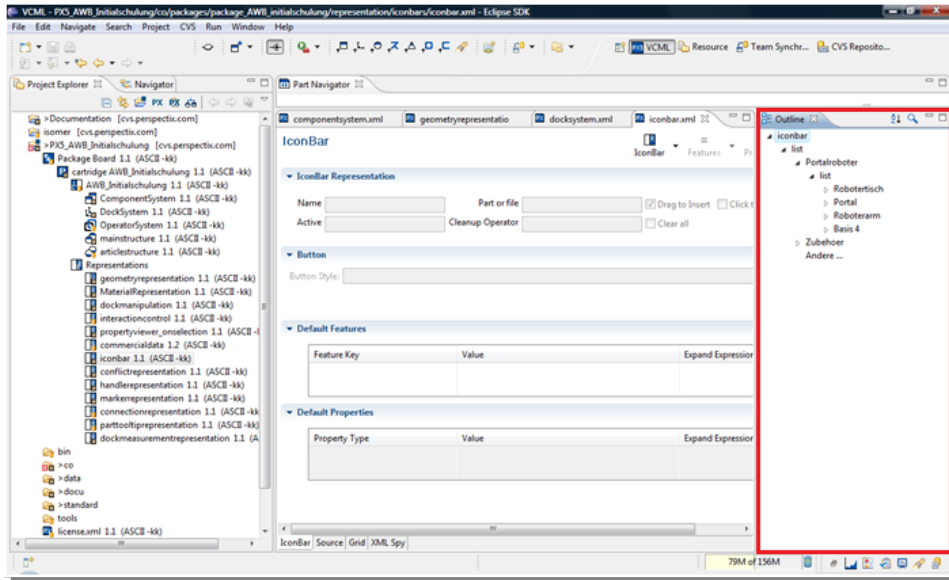
For the robot arms you will only create one icon, even though you created two components for the robot arms, because the robot arms should be accessed together and not separately from the Iconbar. They will be saved in a PAR file that is then assigned to the corresponding icon.

Procedure

1. Open the **Iconbar Editor**.

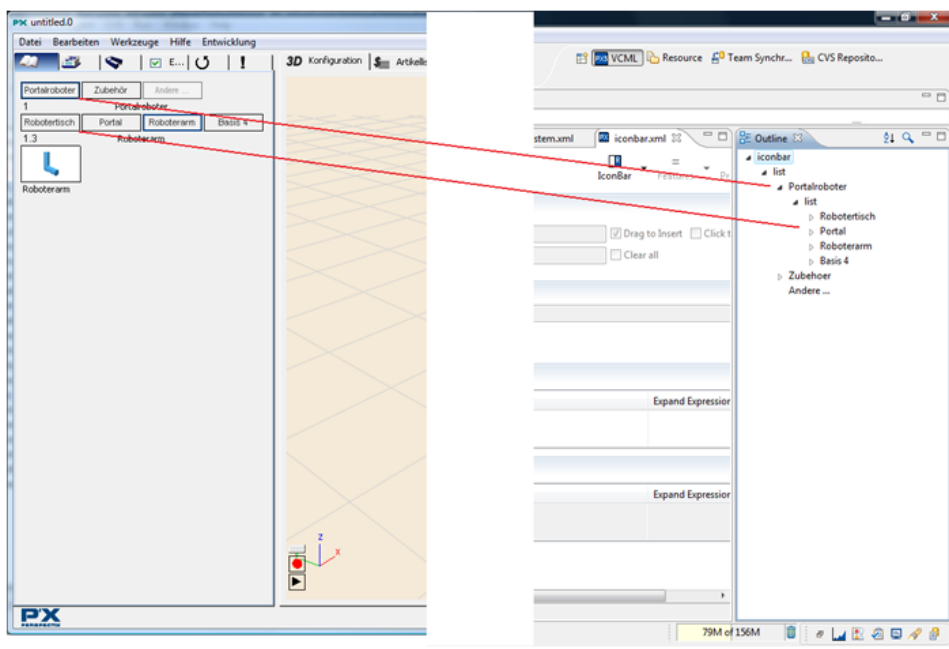


2. Look for the already created **Gantry Robot** and **Supplies** icons in the **Outline** of the **Iconbar Editor**.

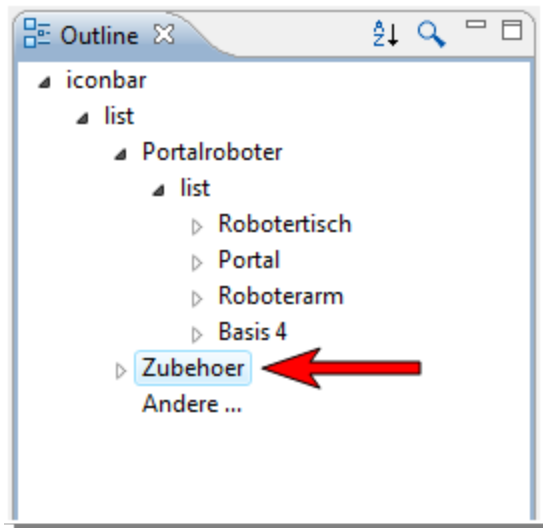


3. Compare the **Iconbar** in the **Configurator** with the **Outline** of the **Iconbar Editor**.

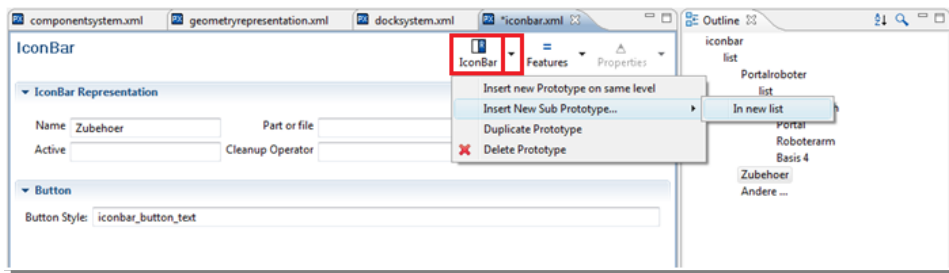
The icons are arranged in the same hierarchic way. A hierarchy level in the **Iconbar** in the **Configurator** corresponds to a hierarchy level in the **Outline** of the **Iconbar Editor**. A row of icons in the **Iconbar** is ordered in the **Outline** as a group of elements with the `list` grouping element.



4. To create an icon for the control panel, select **Supplies** in the **Outline** of the **Iconbar Editor**. A new list and a new prototype for an icon is created under **Supplies**.

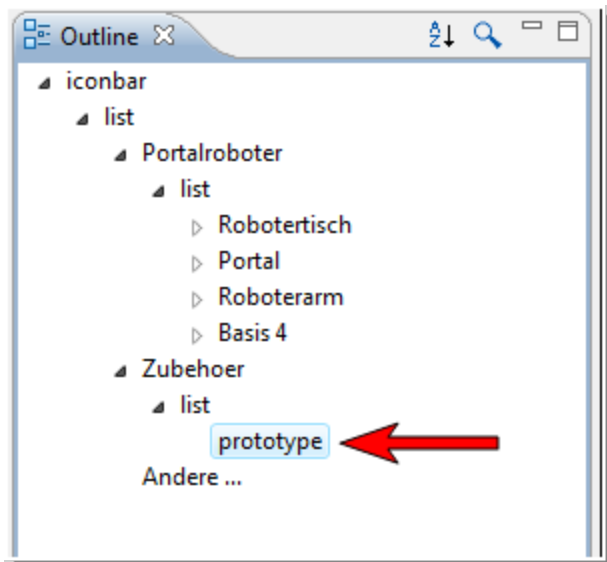


- Click the black arrow next to the **IconBar** button in the **Iconbar Editor** and choose **Insert New Sub Prototype > In new list** from the context menu.



A new list and a new prototype for an icon is created under Supplies in the **Outline** of the **Iconbar Editor**.

- Select **prototype** in the **Outline** of the **Iconbar Editor**.

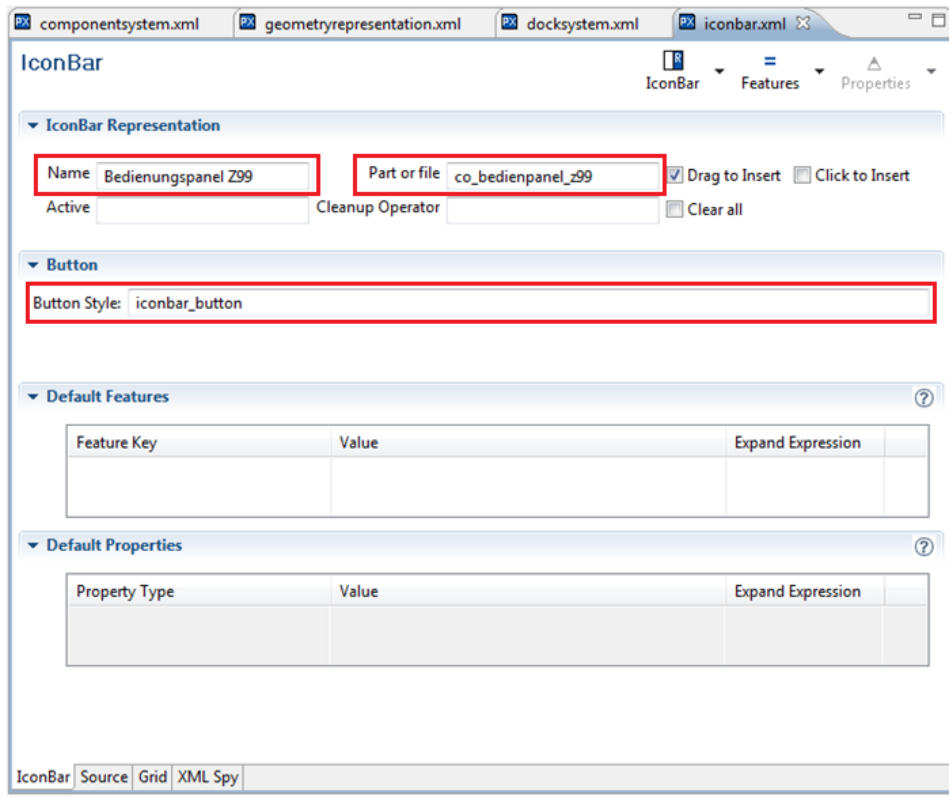


- Type **Control panel 99** in the **Name** field in the **Iconbar Editor** and **co_control_panel_99** in the **Part or file** field or press **CTRL+SPACEBAR** and select the desired entry from the

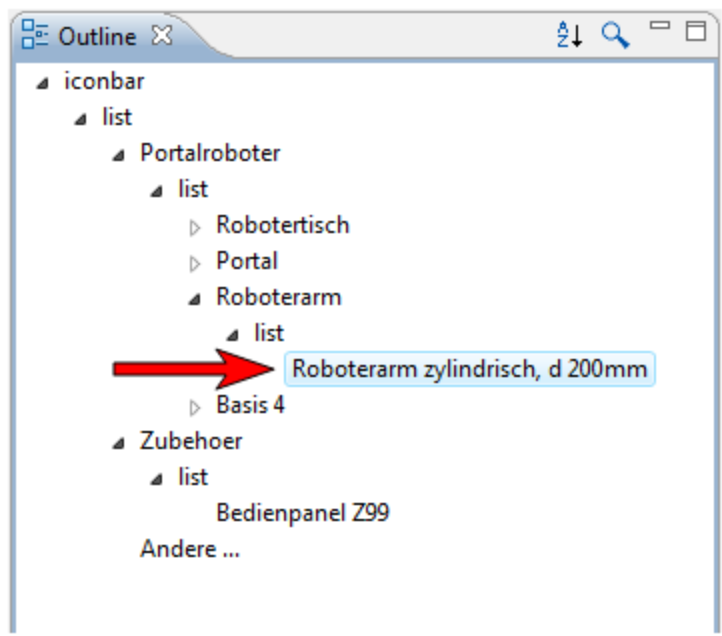
pop-up window.

8. Type **iconbar_button** in the **Button Style** field.

This entry refers to the look of the icon. These entries are saved in another file and can be accessed here.

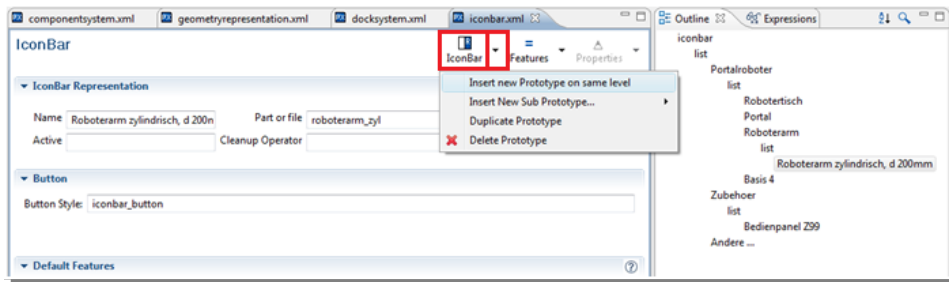


9. Now, create the icon for the robot arm. The robot arms should be accessed together in one icon. To do this, select **Robot arm cylindric** in the **Outline** of the **Iconbar Editor**.

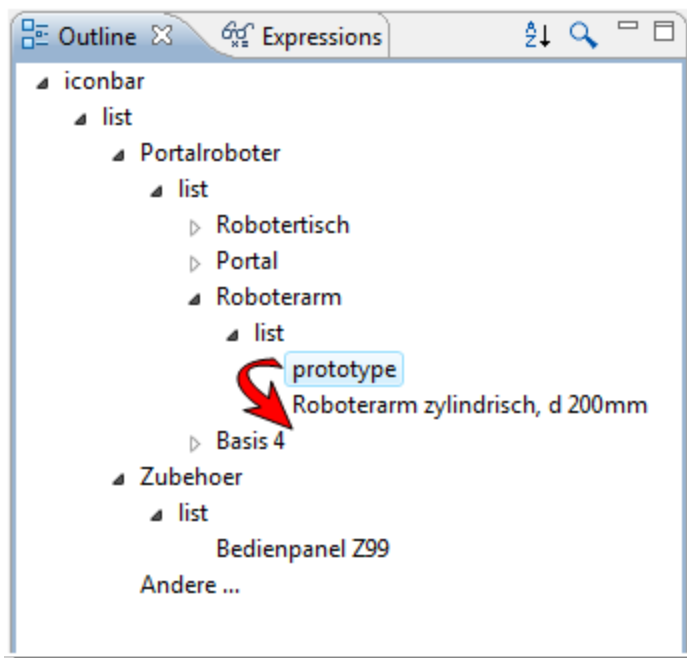


- Click the black arrow next to the **IconBar** button in the **Iconbar Editor** and choose **Insert new Prototype on same level** from the context menu.

An icon on the same level is created.

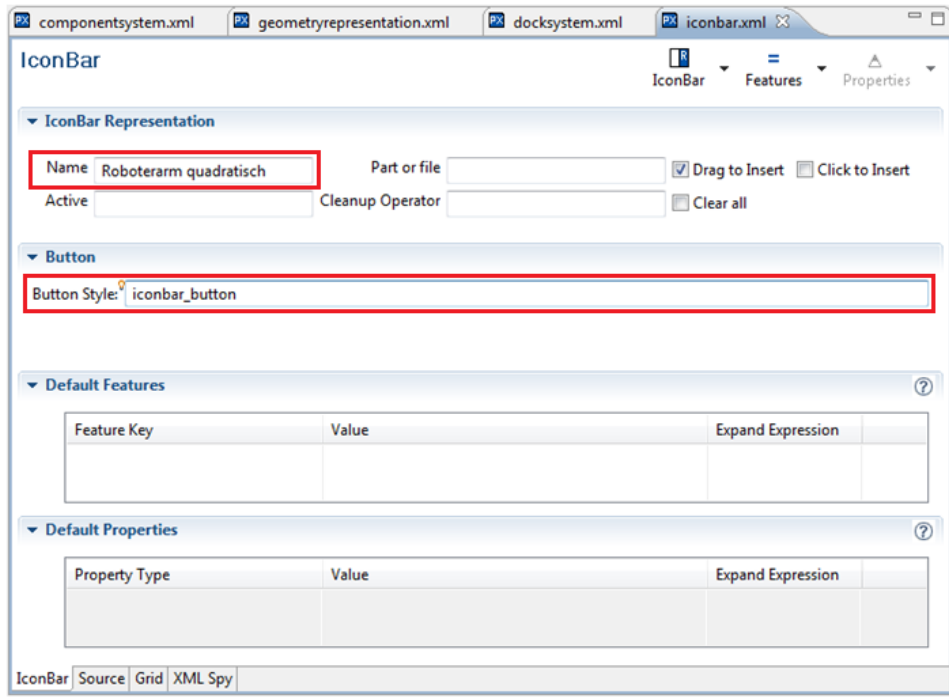


- An icon prototype is created above the Robot arm cylindric entry. You can move the entries in the **Outline** using drag and drop. Click the prototype entry and drag it under the Robot arm cylindric entry. Then select the newly created prototype again.



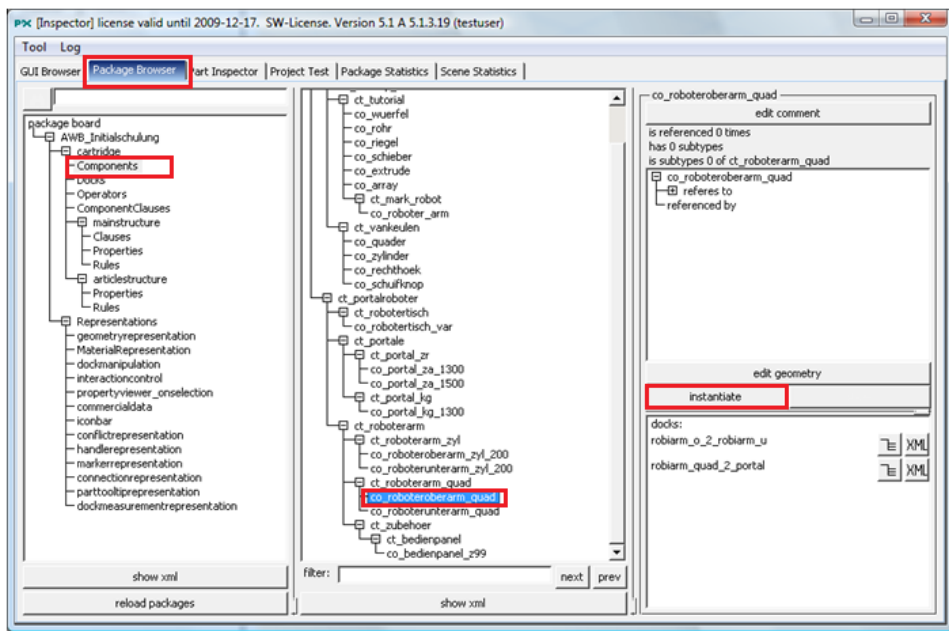
- Type **Robot arm quadratic** in the **Name** field and **iconbar_button** in the **Button Style** field in the **Iconbar Editor**.

In the **Part or file** field, you must enter the name of the file, which contains the two robot arms. However, you must first create this file.



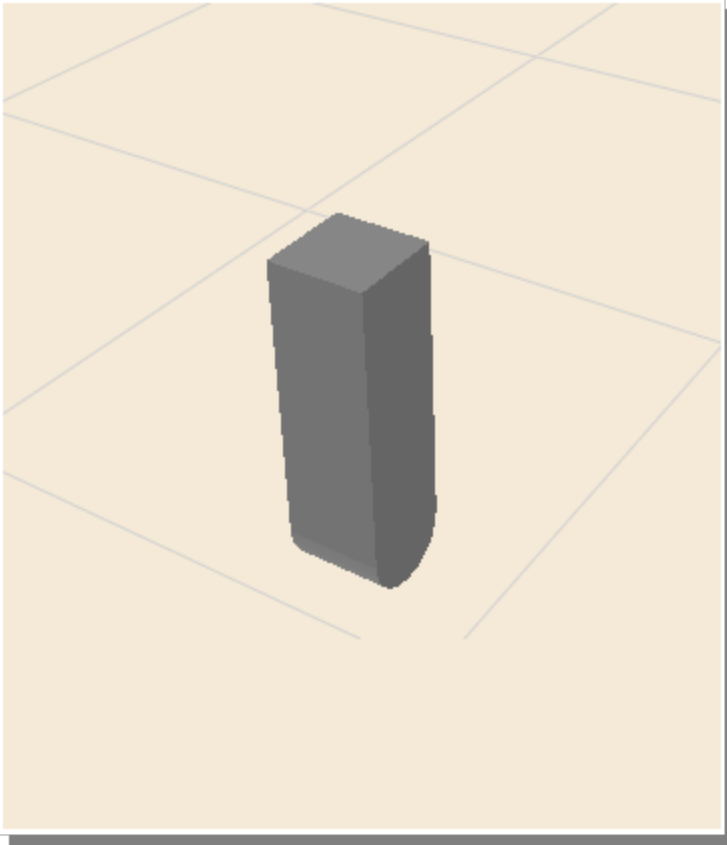
13. Start the **Configurator** by running the `start.bat` file in the **Project Explorer** or via the **Run** menu in Eclipse. If the **Configurator** is already open, press F5 to update it.
14. Press SHIFT+F12 in the **Configurator** to open the **Inspector**.
15. Click the **Package Browser** tab in the **Inspector** and select the `Components` entry.

Note: If you cannot see this entry, click the plus sign in the square on the left side of the entry.

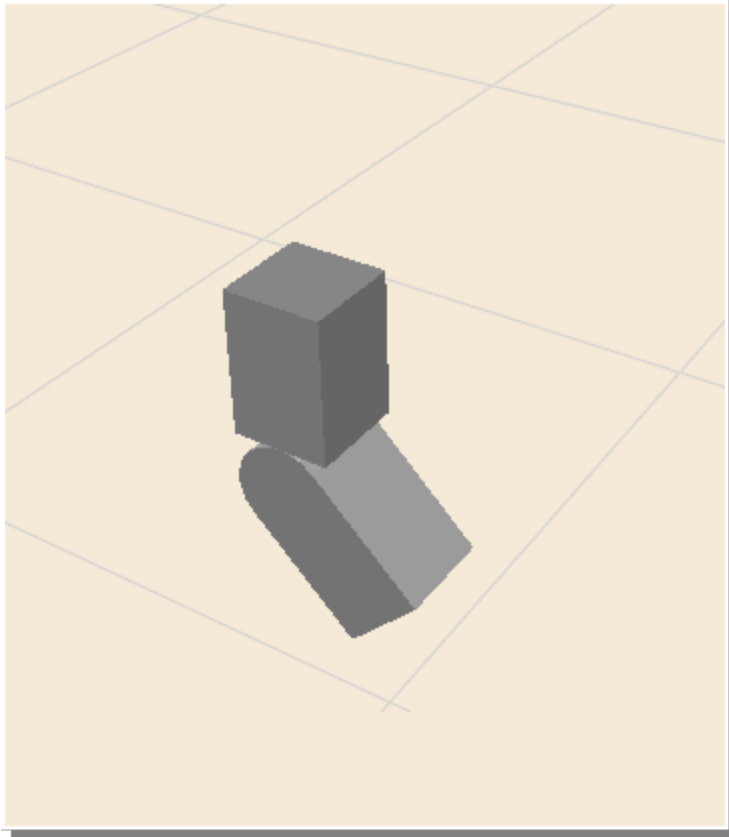


16. Select `co_upper_robotarm_quad` in the middle pane and click the **Instantiate** button. The component, which you already created appears in the **3D Scene** of the **Configurator**.

17. Select `co_lower_robotarm_quad` in the middle pane and click the **instantiate** button to make this component also appear in the **3D Scene**.

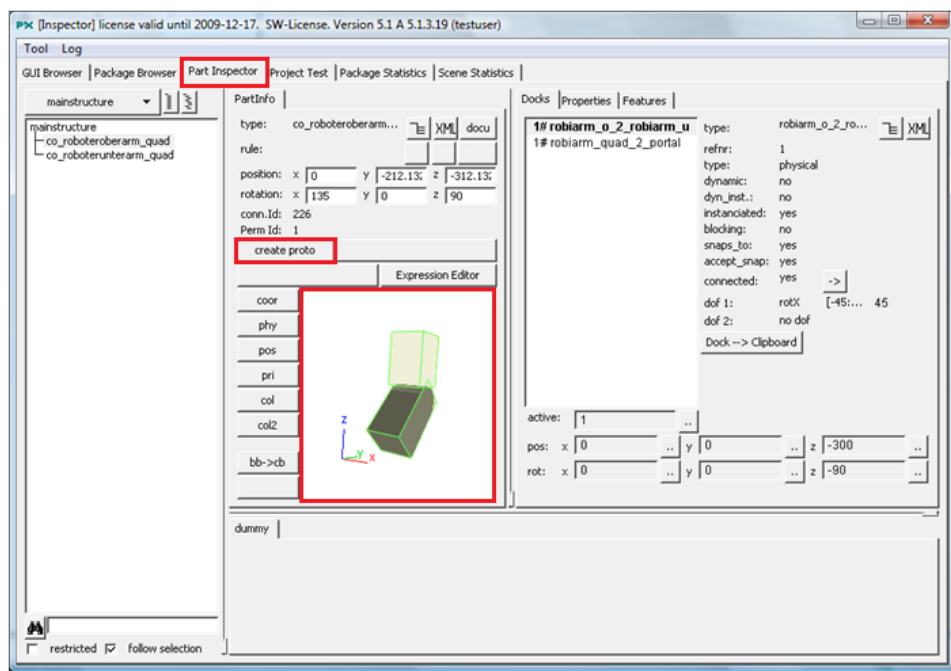


18. Select the lower part in the **3D Scene** and drag it onto the upper part.
The two parts connect at the point where you attached the docks.



19. Press SHIFT+F12 to open the **Inspector**.
20. Click the **Part Inspector** tab and click the middle window underneath the **create proto** button.

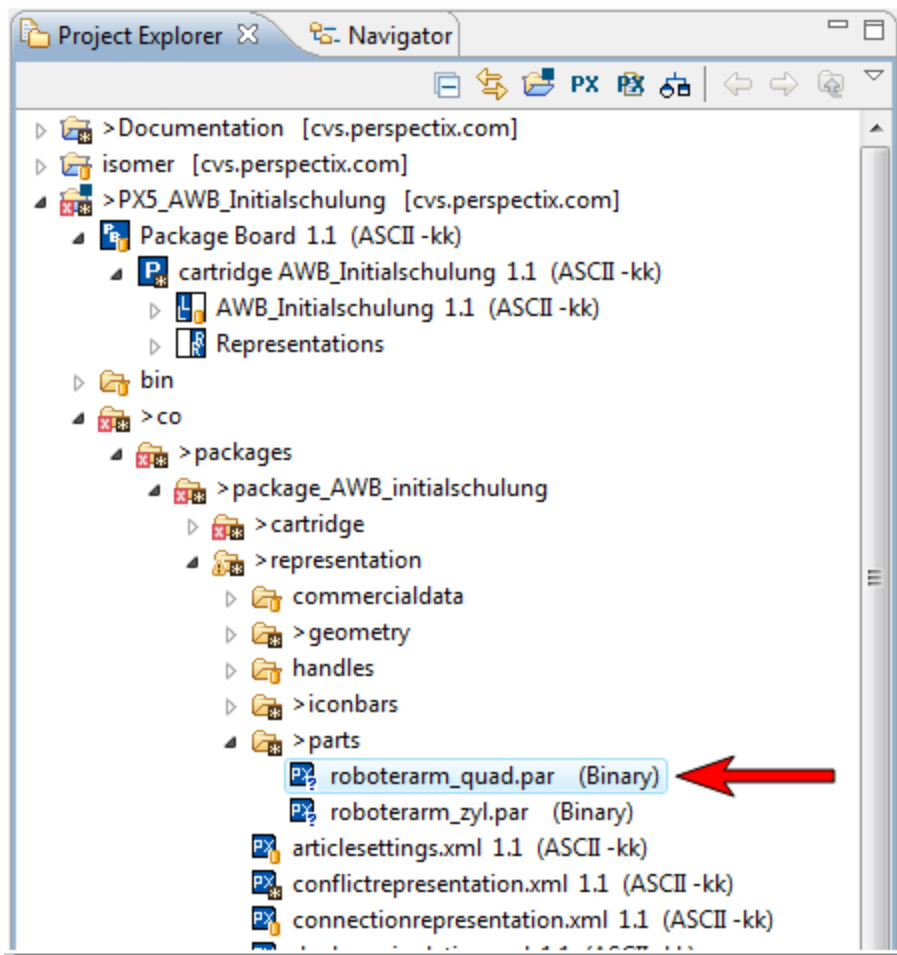
Note: If you do not see the two robot arms, press the SPACEBAR. The two robot arms should appear.



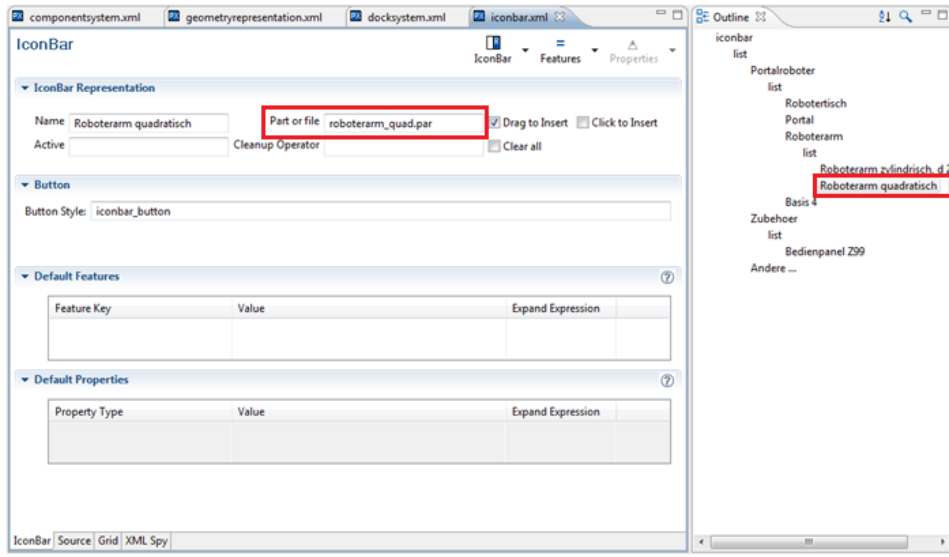
21. Click the **create proto** button and save these two robot arms as a `robotarm_quad.par` file in the `\\PX5_AWB_Tutorial\co\packages\tutorial\representation\parts` directory.

Note: The `.par` ending is necessary, so that this file is recognized as a PAR file. This file will be called from this directory at a later time.

You can see the saved file in the **Project Explorer** in the `\\PX5_AWB_Tutorial\co\packages\tutorial\representation\parts` directory.



22. Select Robot arm quadratic in the **Outline** of the **Iconbar Editor** and type the name of the `robotarm_quad.par` PAR file that you just created in the **Part or file** field in the **Iconbar Editor**. You already defined the name of the icon and the button style.
23. Save the **Iconbar Editor**.



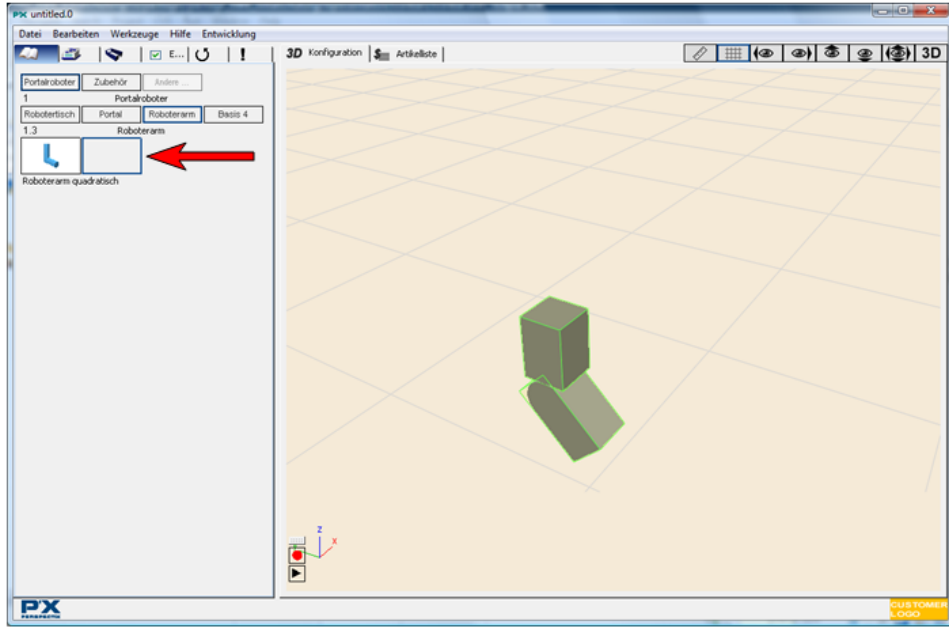
24. Start the **Configurator** again or if it is still open, press F5 to update it.

25. Select the **Gantry Robot** icon in the **Iconbar**.

Other icons appear.

26. Select the **Robot arm** icon in the lower row.

Two more icons appear, one with a round robot arm and the one that you just created. Your newly created icon is missing the image of the robot arm. Your second newly created icon **Control panel 99** under **Supplies** is also missing an image.



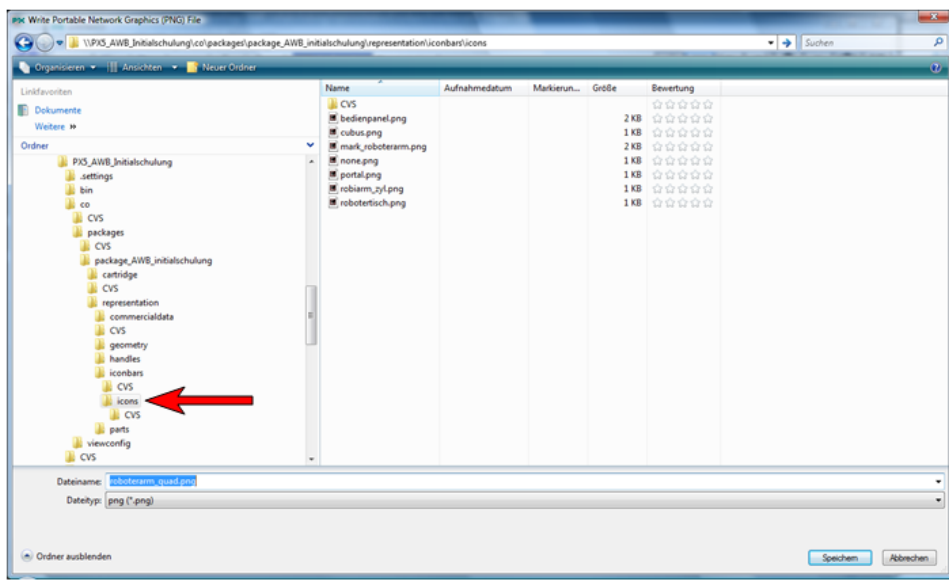
27. Click in the **3D Scene** and press F3 to open the **Icon Generator**. The **Icon Generator** shows the robot arms.

28. Press the SPACEBAR to focus the object.

Tip: Press and hold down the mouse wheel and press SHIFT to navigate in the view.

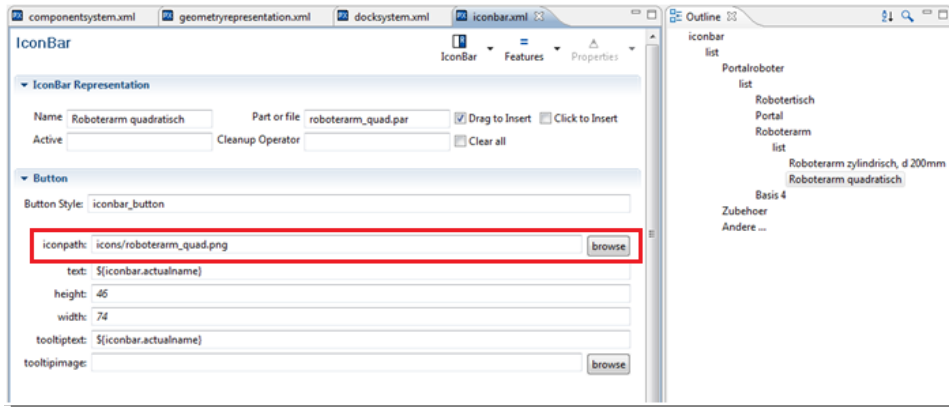


29. Click the **Create Icon** button in the **Icon Generator** to save the image.
30. Select the `\\PX5_AWB_Tutorial\co\packages\tutorial\representation\iconbars\icons` directory in the **Explorer** and save the image as `robotarm_quad.png`.

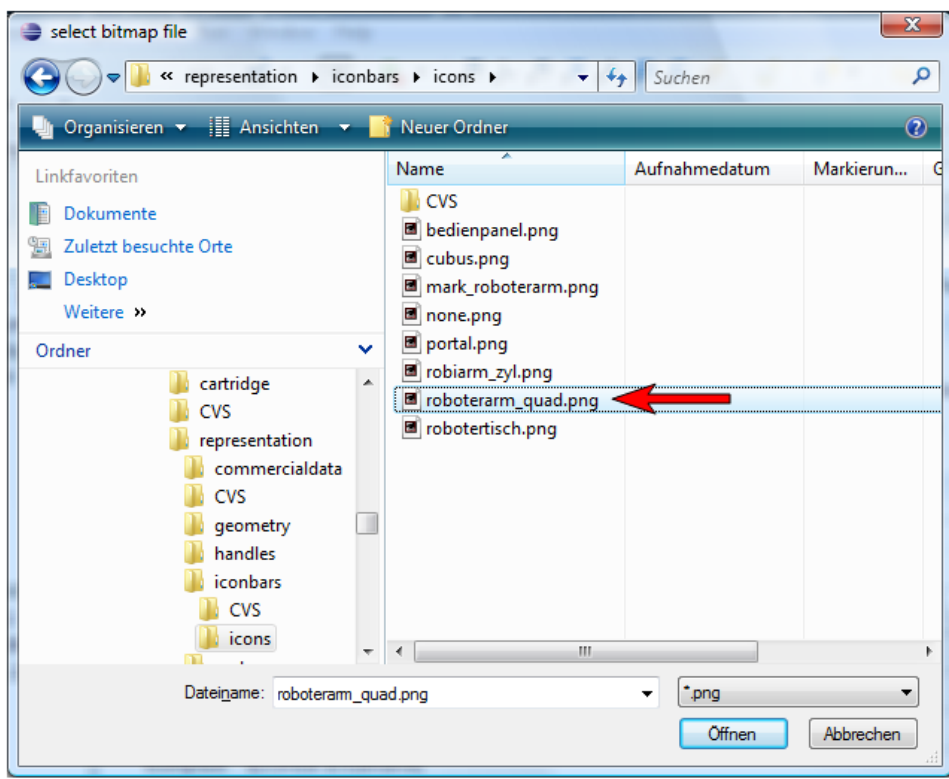


31. Place the mouse cursor in the **Button Style** field in the **Iconbar Editor**, where you already typed `iconbar_button` and press TAB.

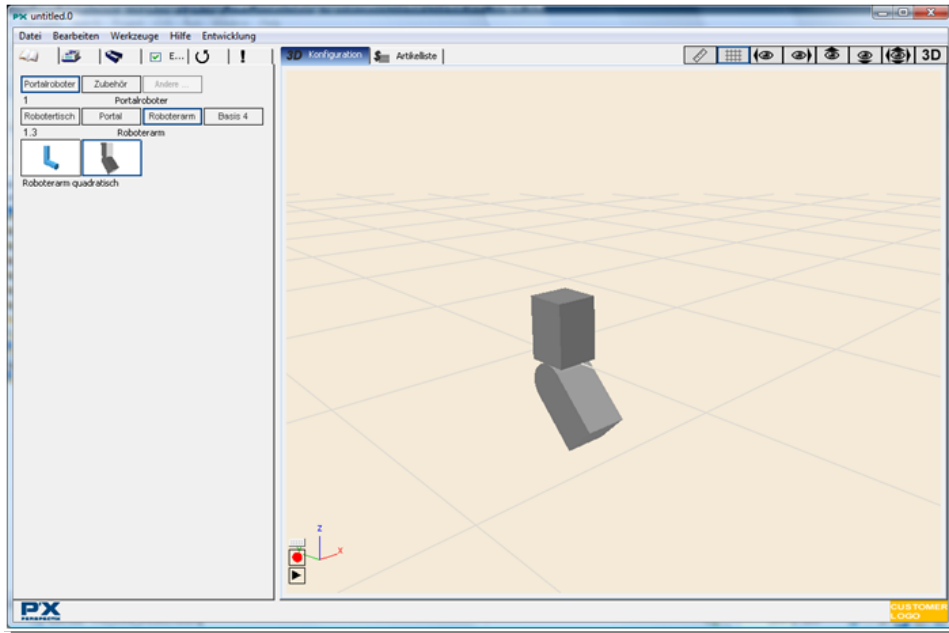
More fields open below the **Button Style** field.



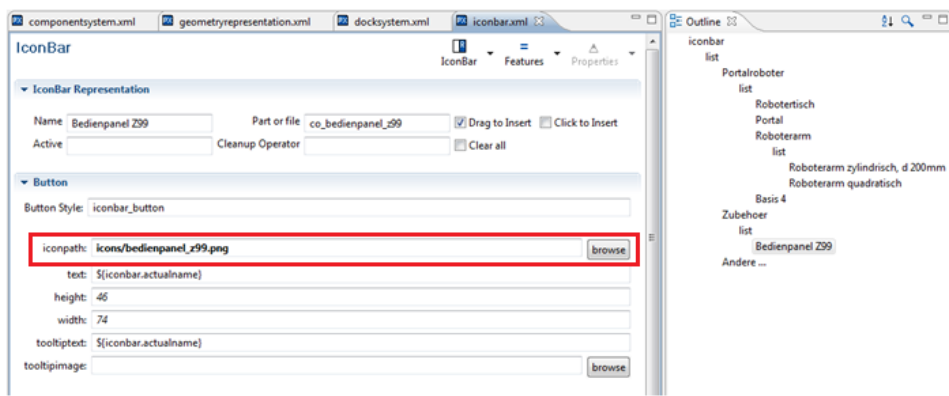
32. Type the path and the name of the image `icons/robotarm_quad.png`, which you just created, in the **iconpath** field or click the **browse** button and select the image in the **Explorer**.



33. Save the **Iconbar Editor**.
34. Start the **Configurator** or, if it is still open, press F5 to update it.
The icon with the image you just created appears in the **Iconbar** of the **Configurator**.
35. Drag the robot arm from the icon into the **3D Scene**. If you don't like the icon, press F3 to open the **Icon Generator** again and create a nicer image.



36. The icon for the **Control panel 99** under **Supplies** does not have an image yet. Assign an image to this icon:
 - a. Delete everything in the **3D Scene** and drag the control panel from the **Control panel 99** icon under **Supplies** into the **3D Scene**.
 - b. Press F3 to open the **Icon Generator**, adjust the image and save it in the same `\\PX5_AWB_Tutorial\co\packages\tutorial\representation\iconbars\icons` directory under `controlpanel_99.png`.
 - c. Select `Control panel 99` in the **Outline** of the **Iconbar**.
 - d. Place the mouse cursor in the **Button Style** field in the **Iconbar Editor**, where you already typed `iconbar_button` and press TAB.
 - e. Type the path and the name of the image `icons/controlpanel_99.png`, which you just created, in the **iconpath** field or click the **browse** button and select the image in the **Explorer**.



37. Save the **Iconbar Editor**.

Result

1. Start the **Configurator** or, if it is still open, press F5 to update it.

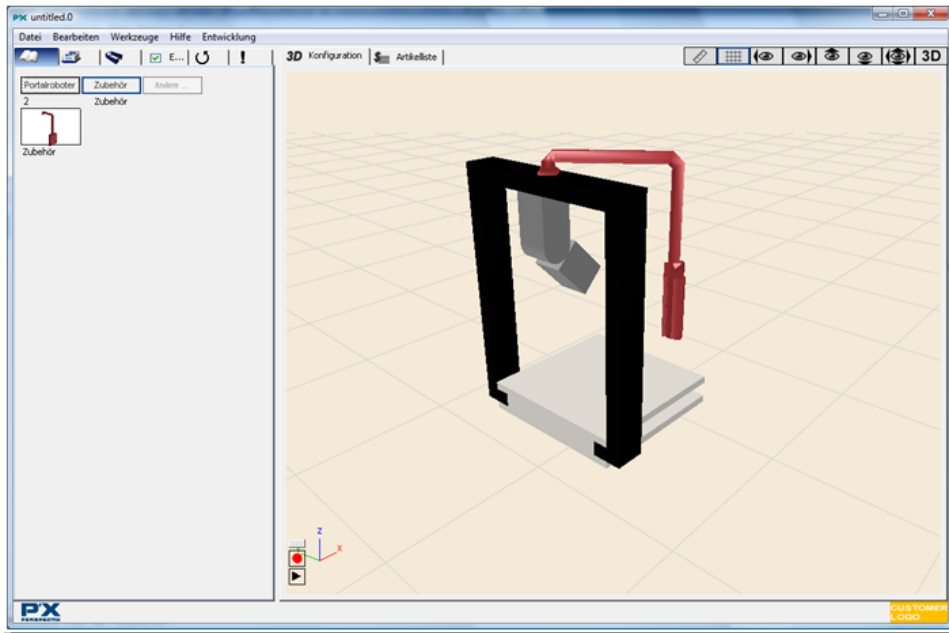
All icons in the **Iconbar** should now have images. Now you can assemble the robot.

2. Drag the table, the gantry, the square robot arm, and the control panel, one after the other, from the **Iconbar** into the **3D Scene** and connect them with each other.

The parts will connect at the points where you created docks.

3. Place the mouse cursor over the parts. When the mouse cursor changes its shape to a hand, you can move or turn the parts.

This is possible at the points where you defined docks with degrees of freedom (DOF).



The square robot arm does not have a color yet. In the next lesson you will learn how to define materials and how to assign them to the geometries.

Now, you can define materials and assign them to geometries.

Lesson 6 - Define Materials and Assign them to Geometries

The square robot arms are missing a material, that is why they are grey in the 3D Scene of the Configurator. If a geometry does not have a material assigned to it, it appears grey.

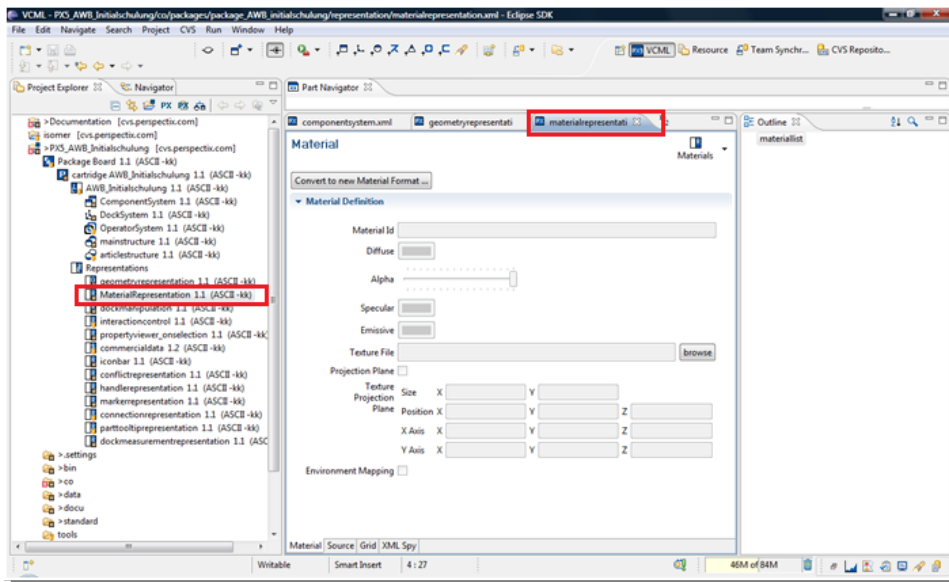
Your Assignment

Assign the following materials to the geometries:

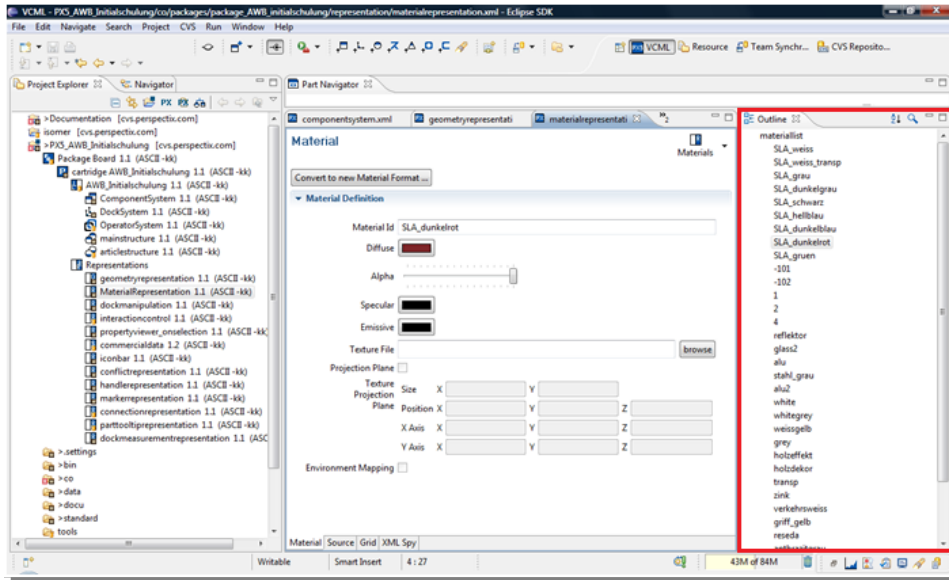
Material	Component (Geometry)
lightblue	co_upper_robotarm_quad
lightblue	co_lower_robotarm_quad
darkred	co_bedinpanel_z99

Procedure

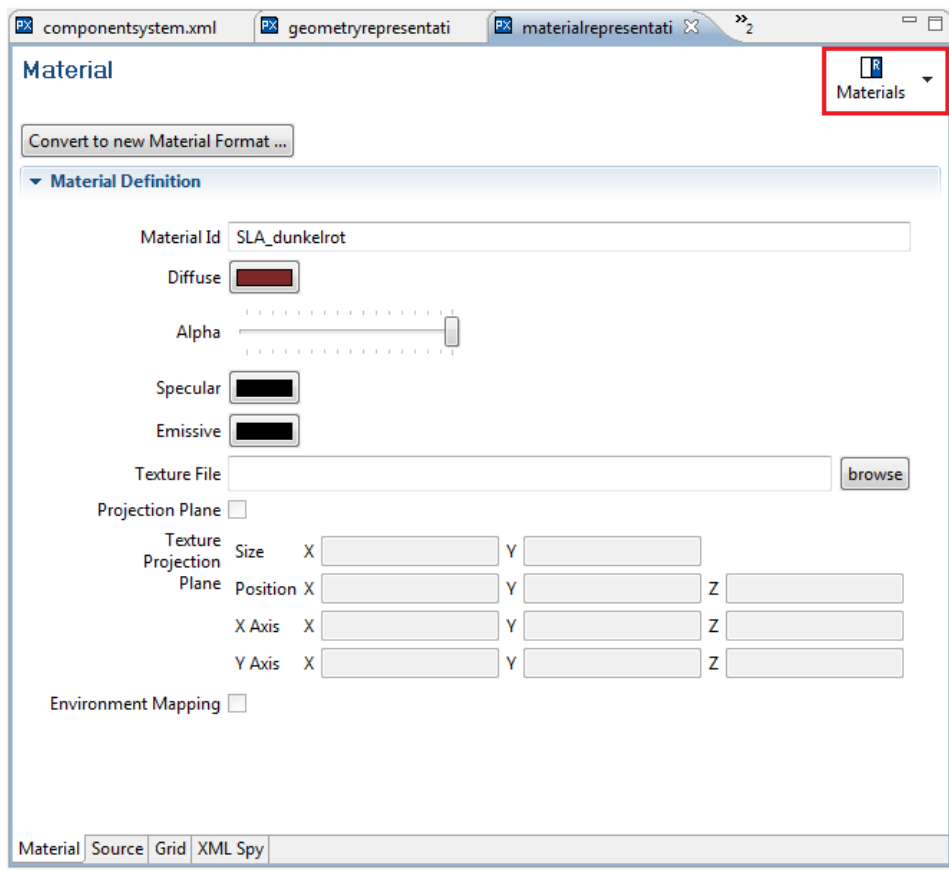
1. Open the **Materialrepresentation Editor**.



2. Take a look at the existing materials, e.g. `darkred` or `lightblue` in the **Outline** of the **Materialrepresentation Editor**. Click the arrow next to the materials to open the list.

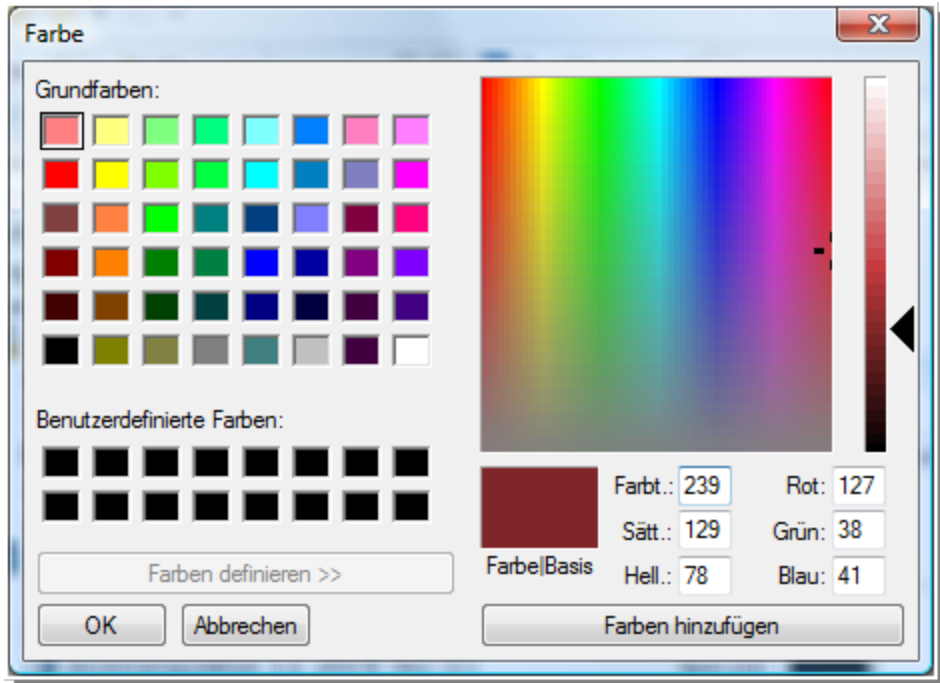


3. Click the **Materials** button in the **Materialrepresentation Editor** to define a new material.

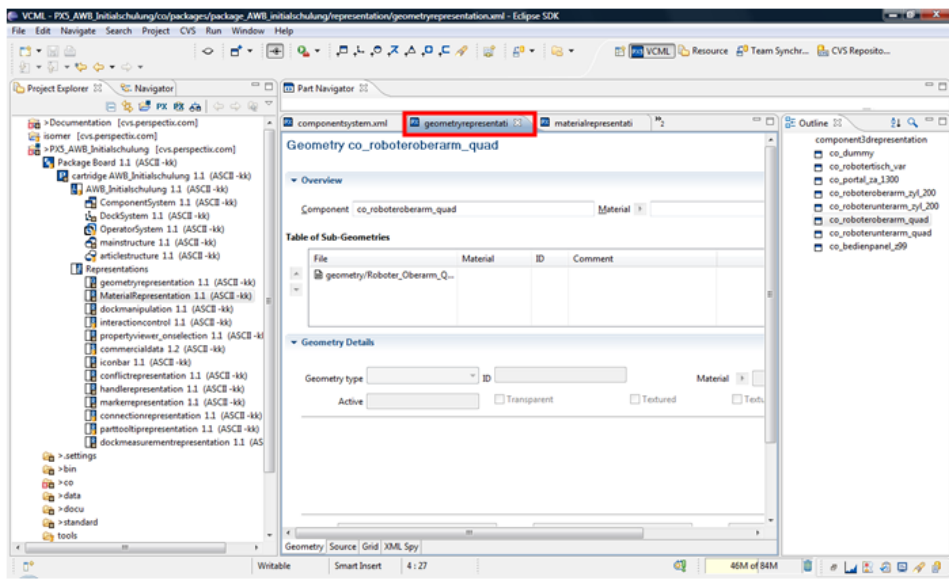


4. Click **Diffuse** to select a color from a color palette.

Note: To define a material requires practice, that is why a few default materials are already defined.



5. Open the **Geometryrepresentation Editor**.



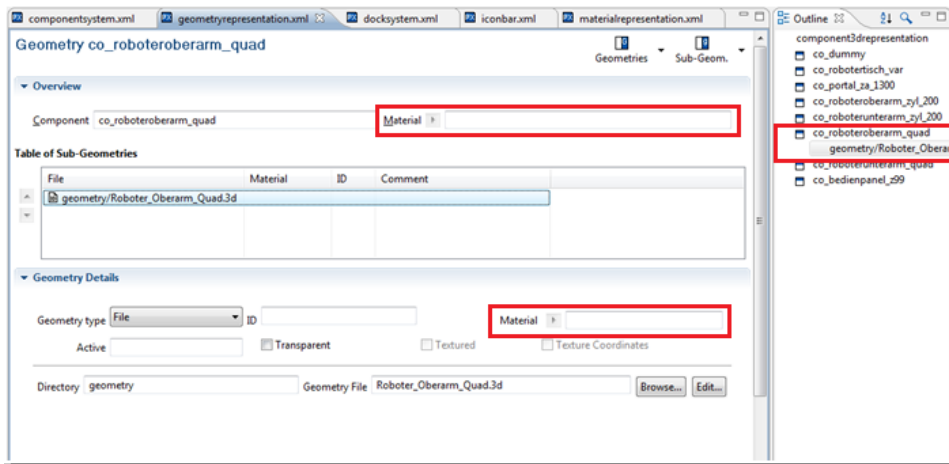
6. Select `co_upper_robotarm_quad` in the **Outline** of the **Geometryrepresentation Editor**.

7. Select `robot_arm_quad_upper.3d` in the **Table of Sub-Geometries** in the **Geometryrepresentation Editor**.

8. There are two ways to assign materials:

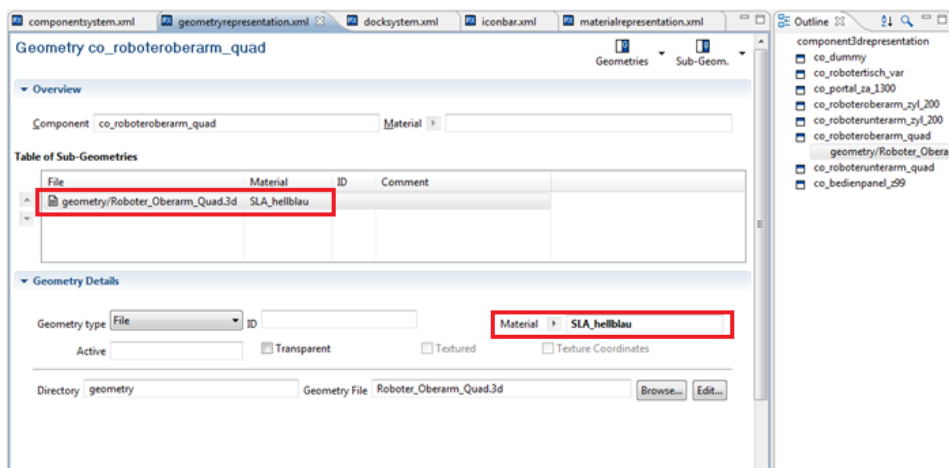
- To assign a material to all subgeometries, enter the material in the **Material** field to the right of the **Component** field.
- To assign a material only to the selected subgeometry from the **Table of Sub-Geometries**, enter the material in the **Material** field to the right of the **Geometry type** field.

Note: If you assigned a material to all geometries and then to a subgeometry again, the material you assigned to the subgeometry will be overwritten by the material you assigned to all geometries.



9. Select `robot_arm_quad_upper.3d` in the **Table of Sub-Geometries** in the **Materialrepresentation Editor**.
10. Type `lightblue` in the **Material** field (to the right of the **Geometry type** field) or press CTRL+SPACEBAR and choose the material from the pop-up window.

Note: You can only assign materials, which are defined in the **Material Representation**.



11. Assign materials to the `co_lower_robotarm_quad` and `co_controlpanel_99` components in the **Geometryrepresentation Editor**:
 - a. Select `co_lower_robotarm_quad` in the **Outline** of the **Geometryrepresentation Editor**.
 - b. Select `robot_arm_quad_lower.3d` in the **Table of Sub-Geometries** in the **Geometryrepresentation Editor** and type `lightblue` in the **Material** field (to the right of the **Geometry type** field).
 - c. Select `co_controlpanel_99` in the **Outline** of the **Geometryrepresentation Editor**.

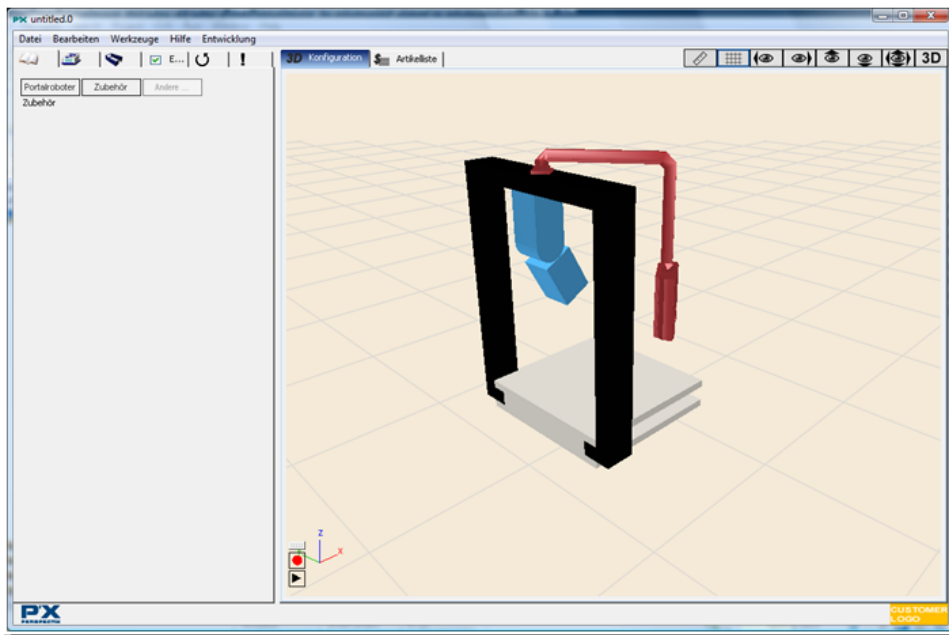
- d. Select `control_panel.3d` in the **Table of Sub-Geometries** in the **Geometryrepresentation Editor** and type `darkred` in the **Material** field (to the right of the **Geometry type** field).

12. Save the **Geometryrepresentation Editor**.

Result

Start the **Configurator** or, if it is still open, press F5 to update it.

Now all parts have a material assigned to them and appear in color in the **3D Scene**.



Now, you can create properties and assign them to geometries.

Lesson 7 - Create Properties and Assign them to Geometries

The length of the robot table should be adjustable. This means that you must first create a property for the adjustable length and then assign this property to a component. The adjustable length will be shown in the 3D Scene.

The robot table should also have a property for the weight. The weight is not shown in the 3D Scene, but is needed in the article list.

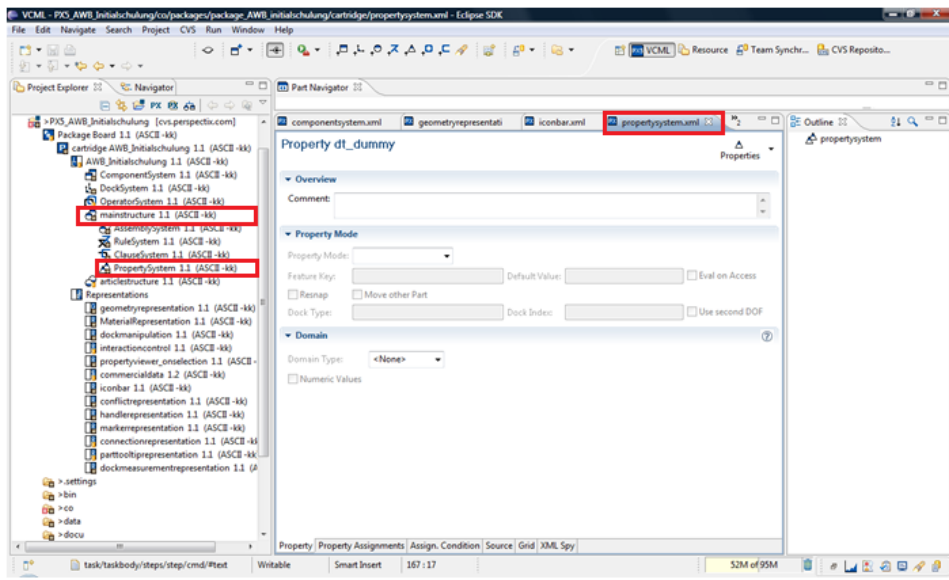
Your Assignment

Assign the following properties to these components:

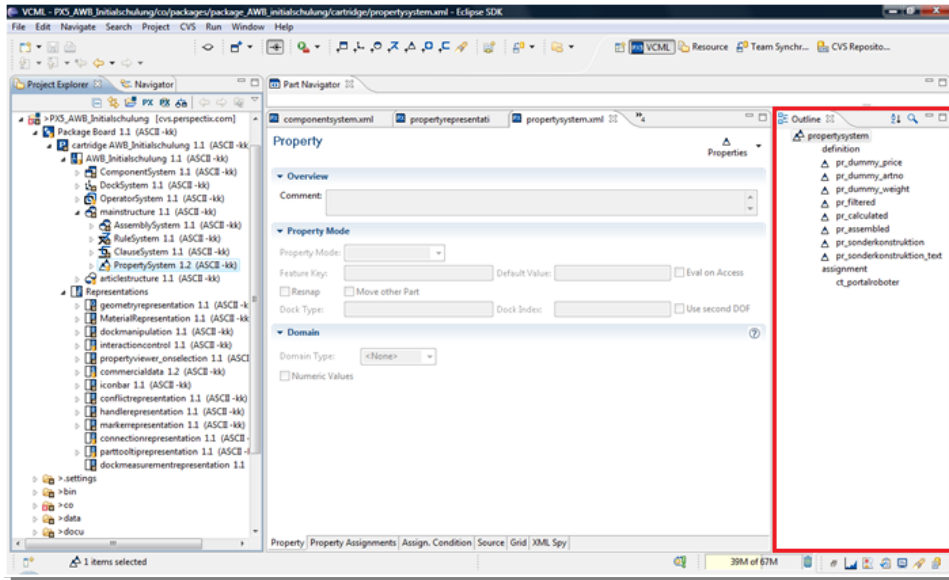
Property	Component
pr_length	co_table_var
pr_weight	co_table_var

Procedure

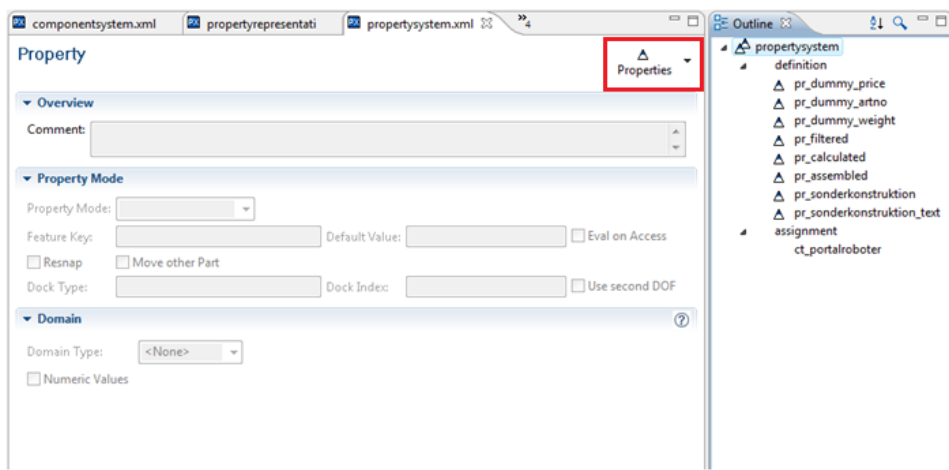
1. Open the **Propertysystem Editor**. The **Property System** is located in the **Project Explorer** under `tutorial/mainstructure`.



2. Take a look at the existing properties in the **Outline** of the **Propertysystem Editor**. Click the arrow to the left of the entry to open the list. The arrow appears when you place the mouse cursor over the entry.



3. Select `propertySystem` in the **Outline** of the **Propertysystem Editor** and click the **Properties** button in the **Propertysystem Editor**.



4. Type `pr_length` in the **Name** field in the **New VCML Property** field and click **Finish**.

Note: The `pr_` prefix is required, because it distinguishes properties from other elements in the code.

Note: Avoid "umlauts" (i.e. vowels, e.g. ü, ö, ä, etc., that do not exist in the English language) in names. Umlauts are interpreted differently on different computer systems.

5. Select the newly created `pr_length` entry in the **Outline** of the **PropertySystem Editor**.

6. Select `feature` in the **Property Mode** list box in the **PropertySystem Editor**.

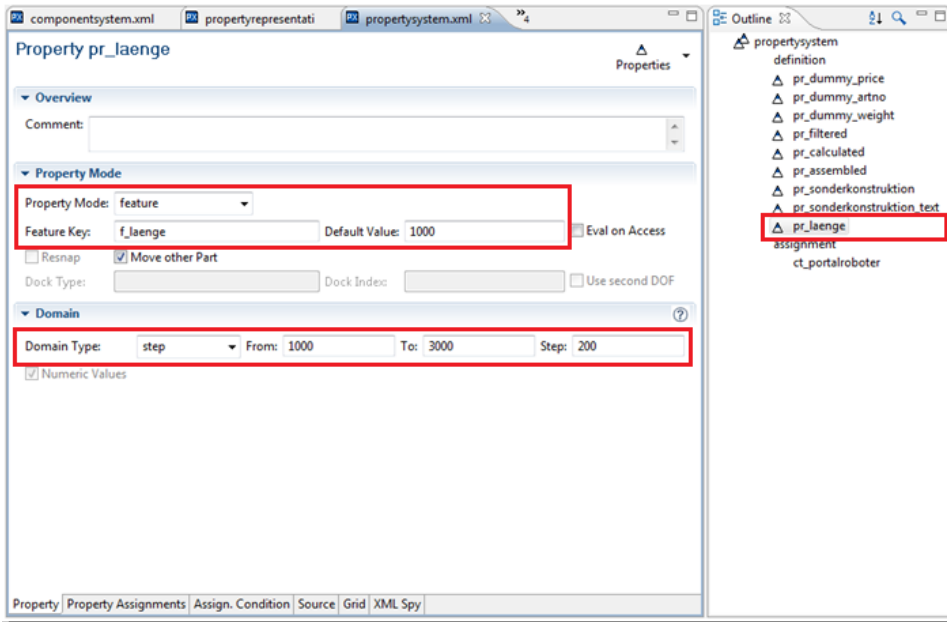
The `f_length` entry in the **Feature Key** field has been created. The `pr_` prefix of the property was replaced by the `f_` prefix. This entry will later be used as a parameter.

7. Type `1000` in the **Default Value** field. The robot table should have the length 1000 as the starting value.

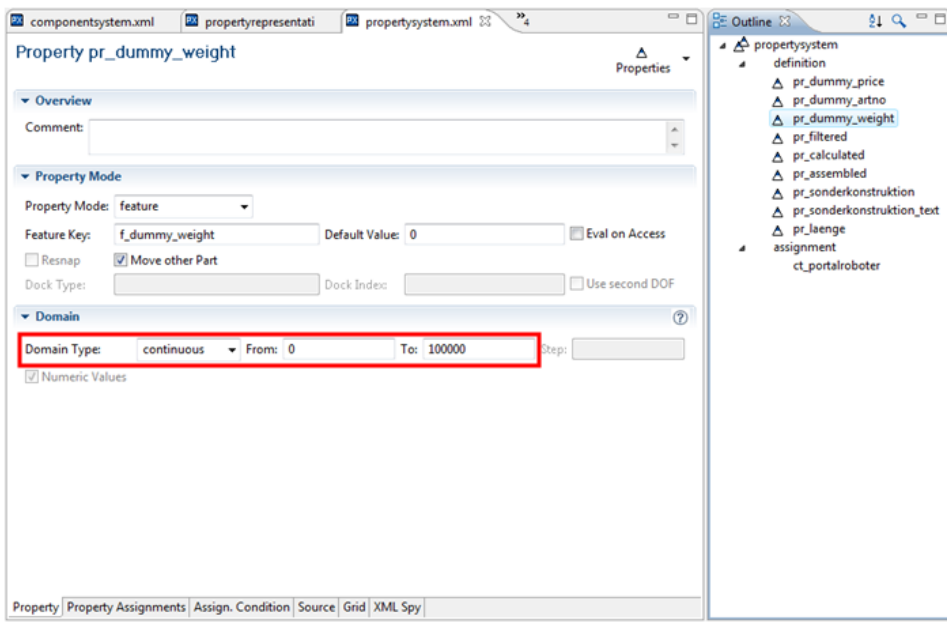
8. Select `step` in the **Domain Type** list box and write `1000` in the **From** field, `3000` in the **To** field and `200` in the **Step** field.

This means that the length of the robot table is a minimum of 1000 units and that it can be increased or decreased from 1000 to 3000 units in steps of 200 units.

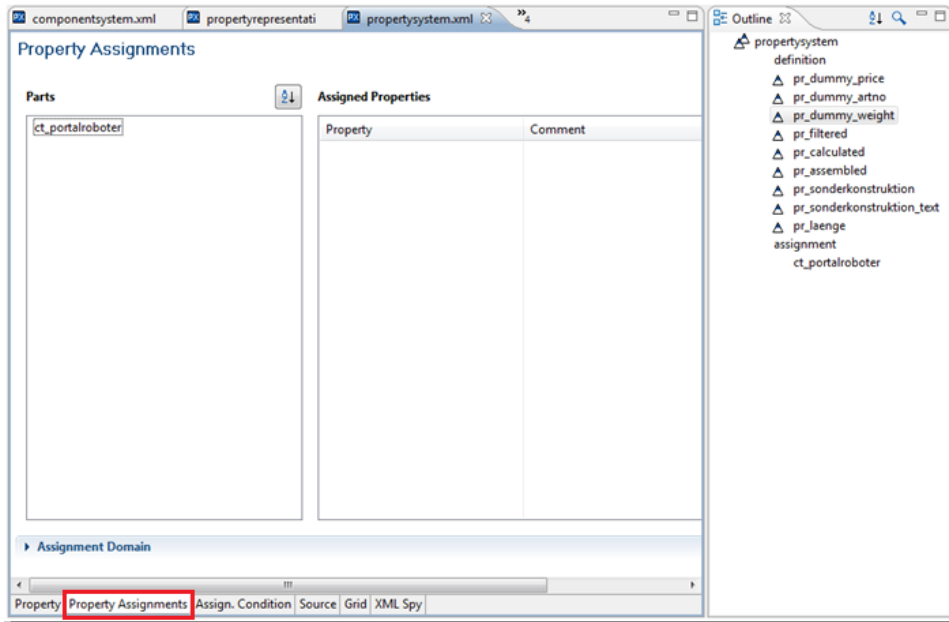
9. Save the **PropertySystem Editor**.



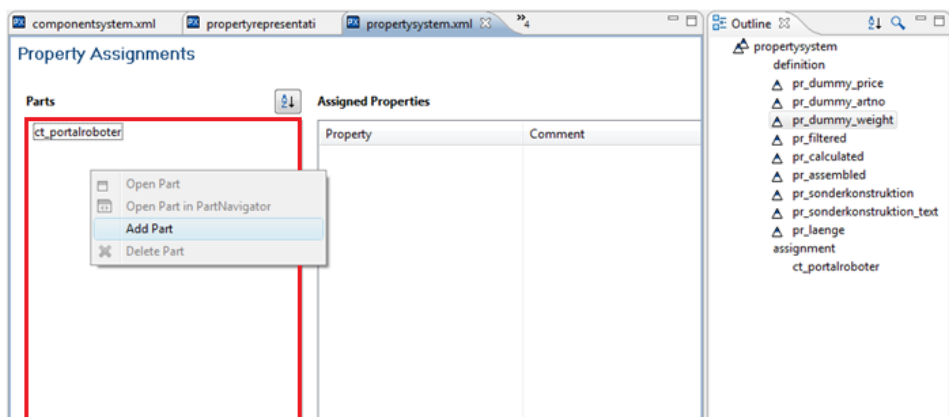
- Compare your newly created `pr_length` property with the existing `pr_weight` property. The difference is that `continuous` is defined in the **Domain Type** field. This means that the value can be changed continuously from 0 to 10000.



- Now you must assign your new property to the correct component. To do this, open the **Property Assignments** page in the **Propertysystem Editor**.

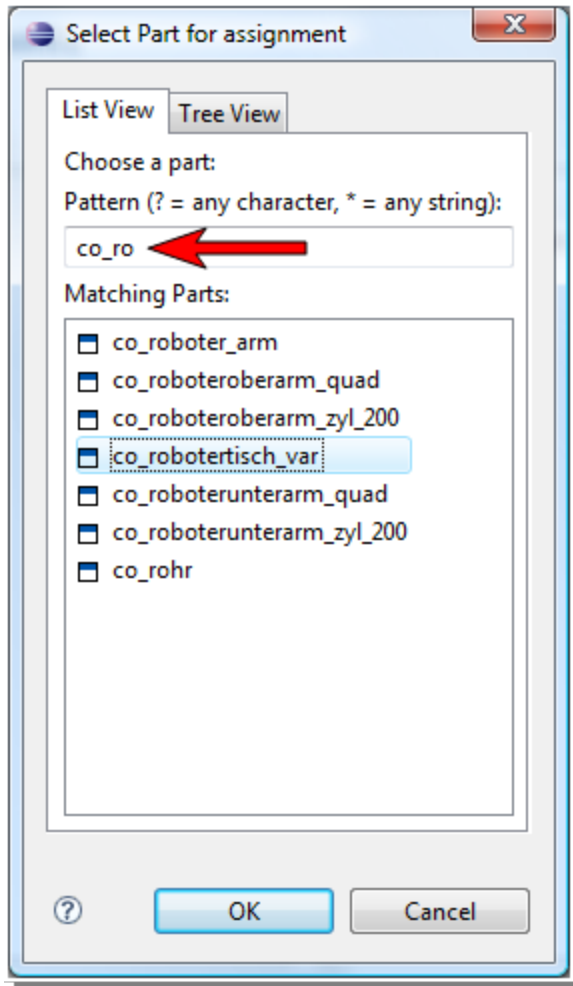


12. Right-click in the **Parts** pane in the **Property System Editor** and choose **Add Part** from the context menu.



13. Select the `co_table_var` component in the **Select Part for assignment** window and click **OK**.

Tip: Type the first letters of the component in the **Choose a part** field to find the component faster.

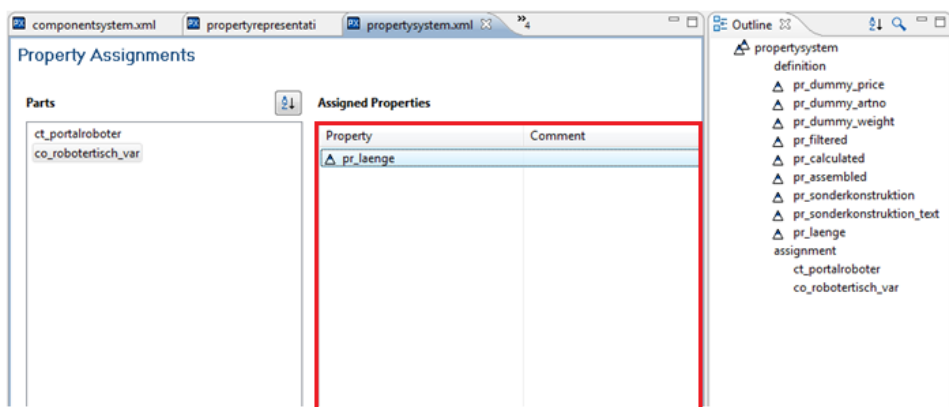


A row is created in the **Assigned Properties** table.

14. Type your newly created **pr_length** property in the new row or press CTRL+SPACEBAR and choose **pr_length** from the pop-up window.

Now the **pr_length** property will be assigned to your **co_table_var** component.

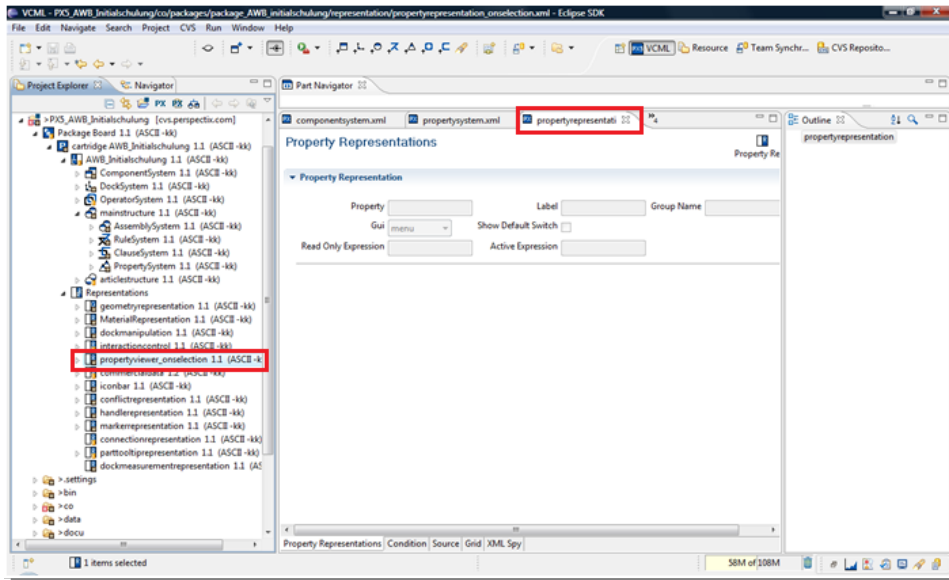
15. Save the **Propertysystem Editor**.



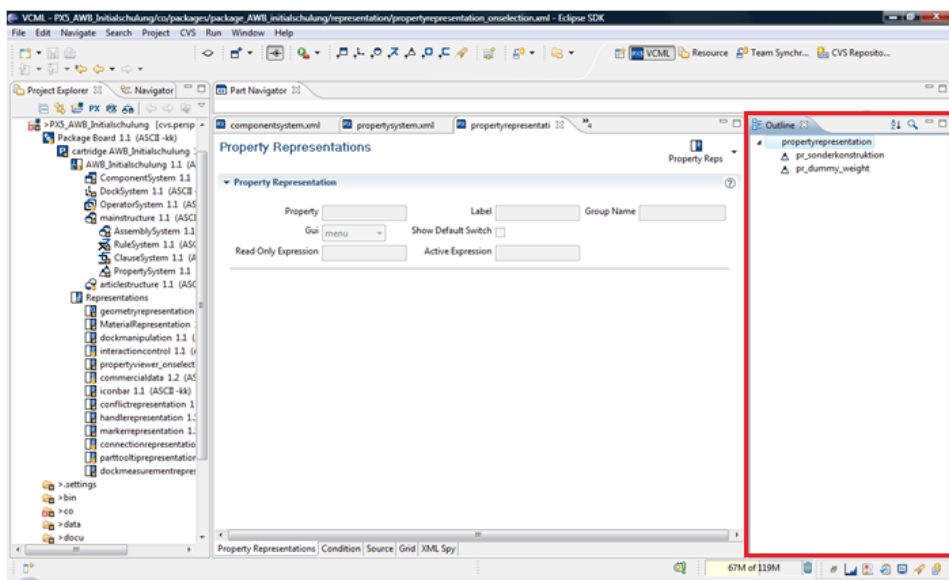
Note: If you want to assign more properties to a component, right-click in the **Assigned Properties** table and choose **New Assignment** from the context menu. A new row will be created, where you can enter another property.

Now you must create a slider in the **Iconbar** in the **Configurator**, with which you can adjust the values of the `pr_length` property. The values should be adjustable from 1000 to 3000 units in steps of 200 units.

16. Select `propertyviewer_onselection` in the **Project Explorer** to open the **Propertyrepresentation Editor**.



17. Take a look at the existing representations of the properties in the **Outline** of the **Propertyrepresentation Editor**.



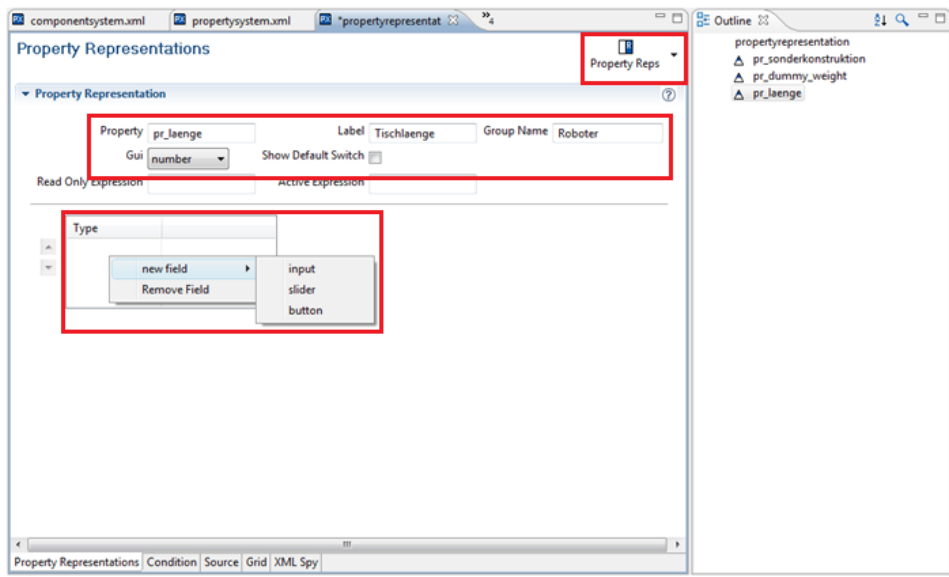
18. Click the **Property Reps** button in the **Propertyrepresentation Editor** to create a representation of the property.

19. Type `pr_length`, the property for which you want to create a new representation, in the **Property** field or press CTRL+SPACEBAR and choose `pr_length` from the pop-up window.
20. Type `Table length` in the **Label** field and `Robot` in the **Group Name** field.
21. Select `number` in the **Gui** list box.

The **Type** field underneath opens. Here you can define the GUI element type, i.e. the way that the interaction element for the property on the user interface looks.

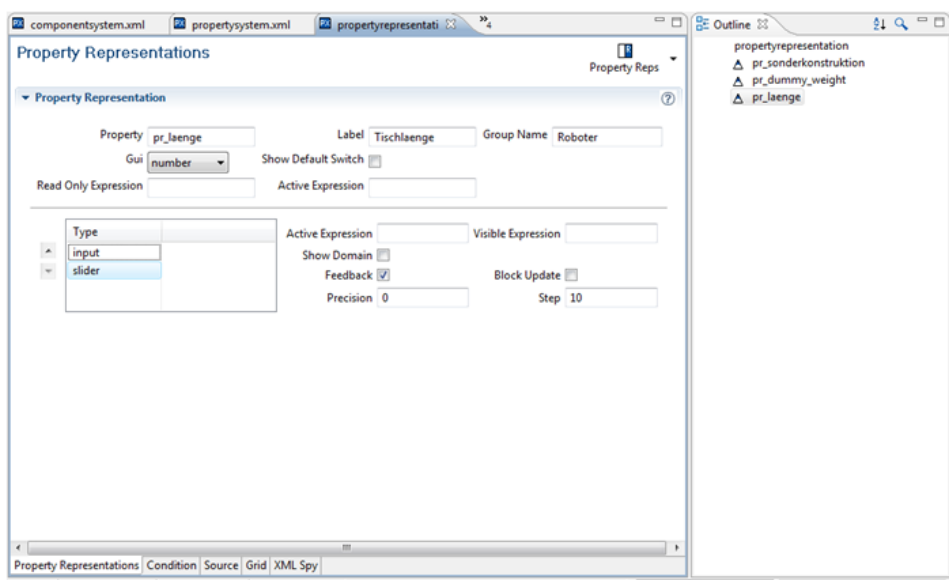
22. Right-click in the **Type** field and choose **new field > input** from the context menu.
23. Right-click again in the **Type** field and choose **new field > slider** from the context menu.

Now you defined an input field and a slider to adjust the values for the `pr_length` property.



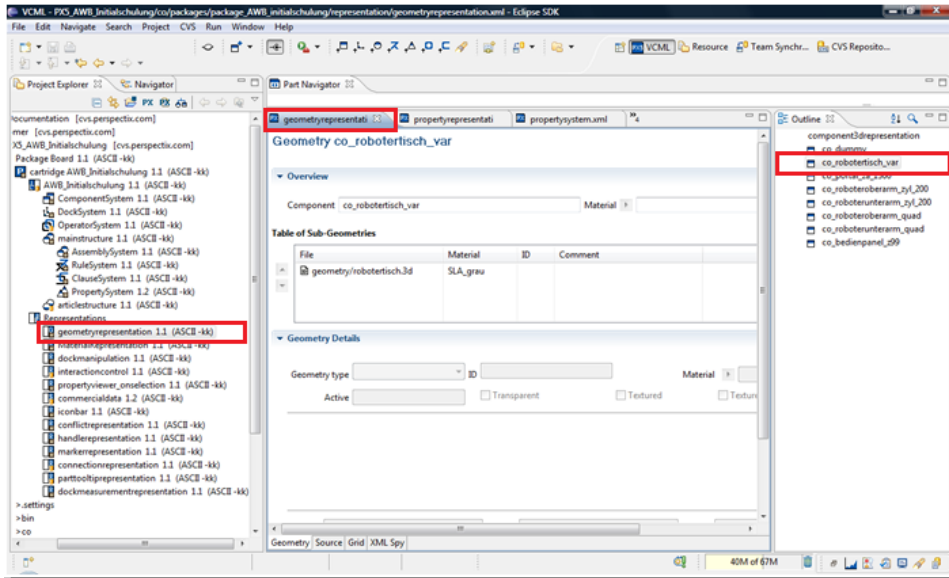
24. Save the **Propertyrepresentation Editor**.

If you have done everything right, your **Propertyrepresentation Editor** should look like in the following figure.



Now the new `pr_length` property should become active in the geometry of the `co_table_var` component. This means that the length of the robot table should be adjustable.

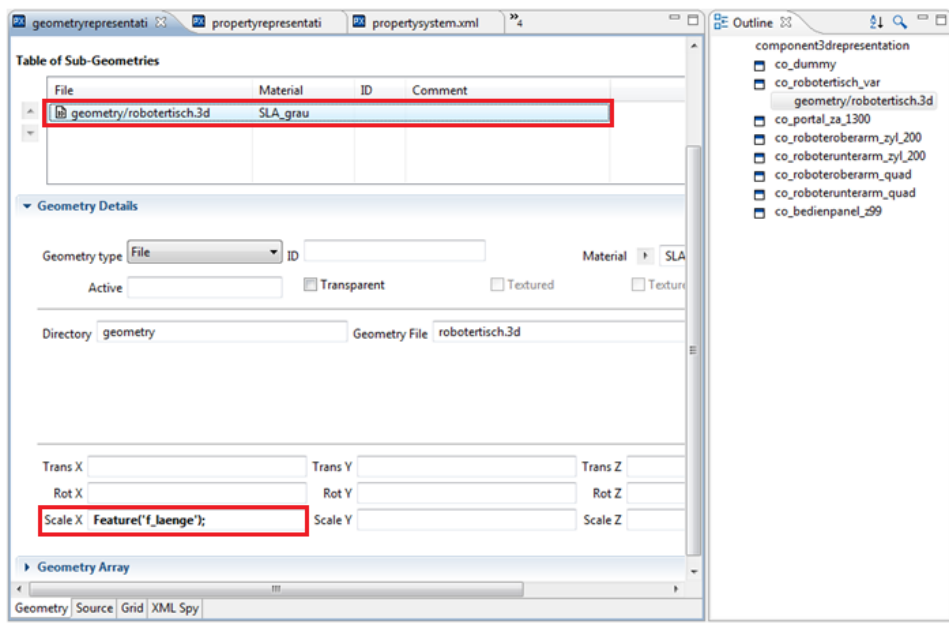
25. Open the **Geometryrepresentation Editor** and select `co_table_var` in the **Outline**.



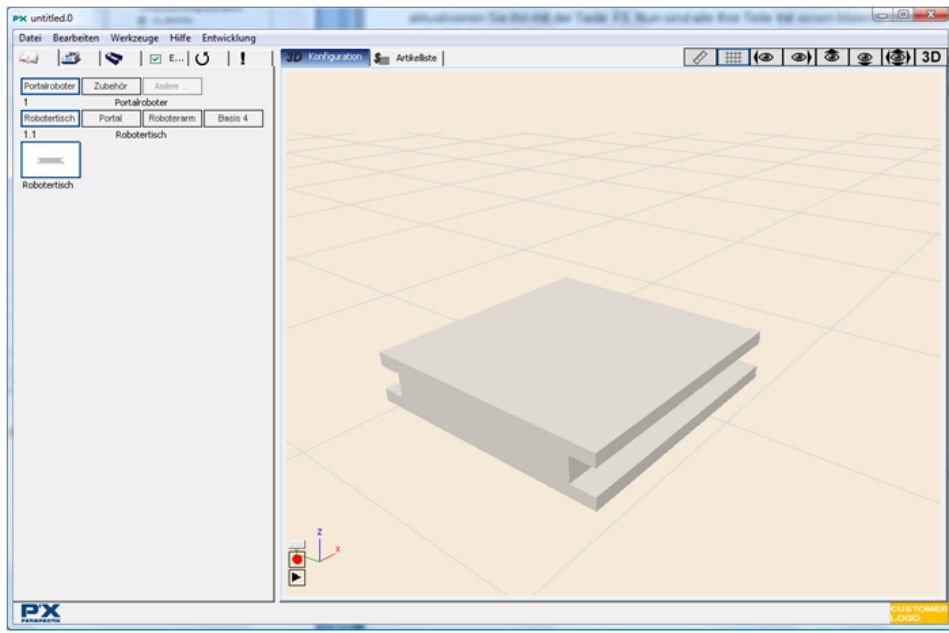
26. Select `robot_table.3d` in the **Table of Sub-Geometries** in the **Geometryrepresentation Editor** and replace `1000` with `Feature('f_length')` in the **Scale X** field.

Note: Note the apostrophe in the parentheses and the semicolon at the end of the entry. `Feature()` is a function with which the properties can be called. When you check in the Property System, you will notice that the `feature` property mode has been chosen for the `pr_length` property.

27. Save the **Geometryrepresentation Editor**.



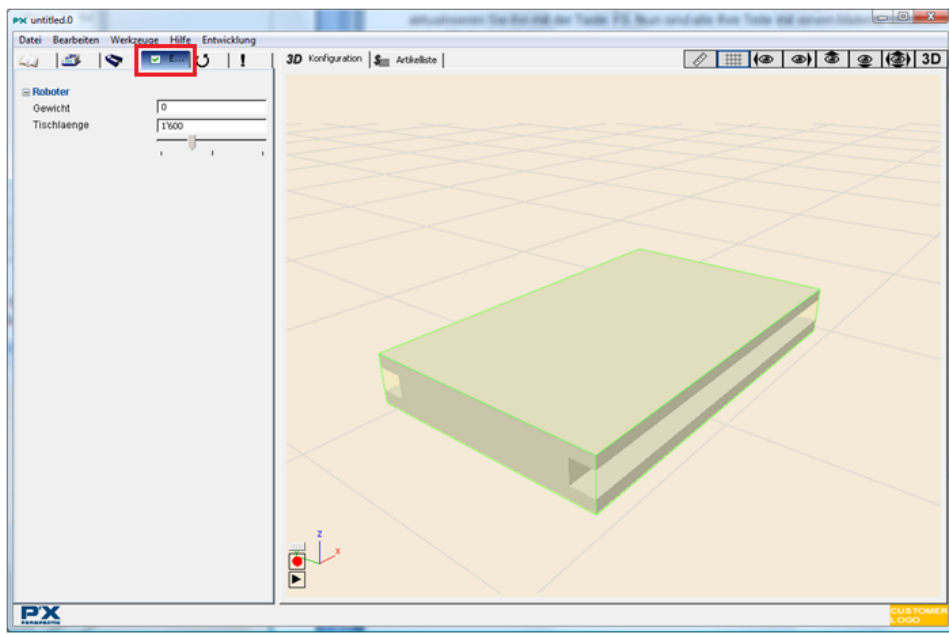
28. Start the **Configurator** or, if it is still open, press F5 to update it.
29. Drag the robot table from the **Iconbar** into the **3D Scene**.



30. Select the robot table in the **3D Scene** and click the **Properties** tab.

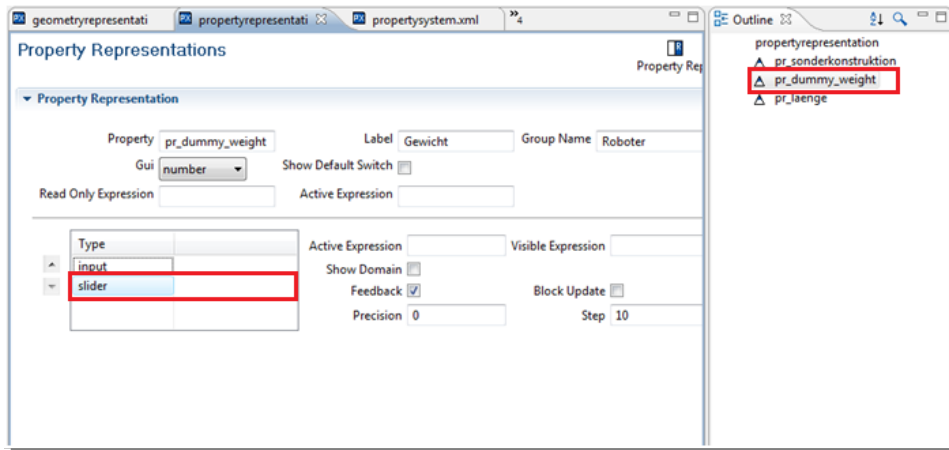
As you can see, the table length can be adjusted with a slider. If you cannot see these two properties, click the plus sign in the square to the left of **Robot**. You can also assign a weight to the robot table, which has no impact in the **3D Scene**, but is listed in the article list.

Note: If the robot table is not selected or if you delete it from the **3D Scene**, the input elements in the **Properties** tab will disappear.



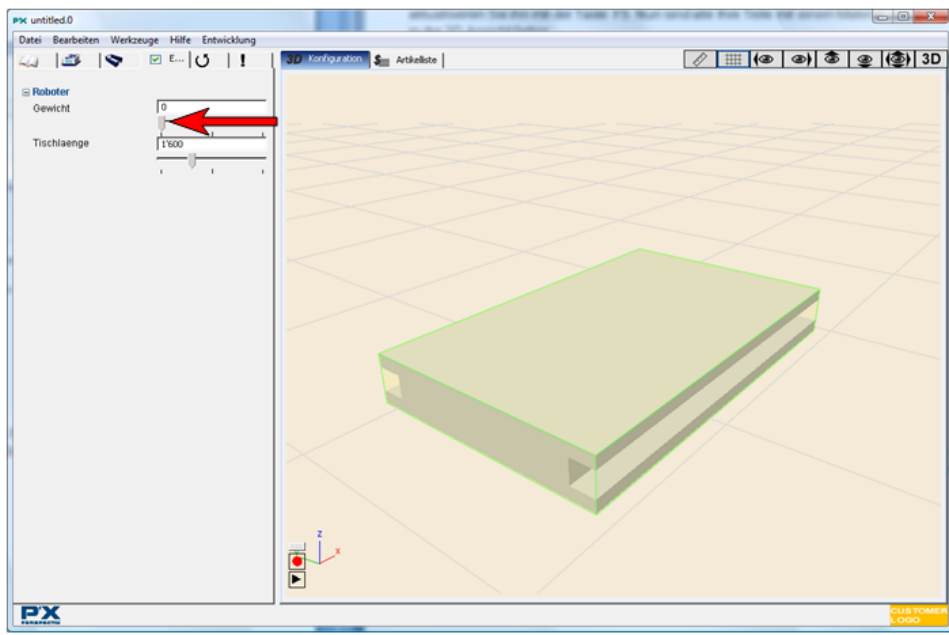
31. Open the **Propertyrepresentation Editor** again. To do so, select `propertyviewer_onselection` in the **Project Explorer**.

32. Select `pr_weight` in the **Outline**.
33. Right-click in the **Type** field in the **Propertysystem Editor** and choose **new field > slider** from the context menu.
34. Save the **Propertyrepresentation Editor**.



35. Start the **Configurator** or, if it is still open, press F5 to update it.
36. Drag the robot table from the **Iconbar** to the **3D Scene**.
37. Click the **Properties** tab and then click the robot table in the **3D Scene**.

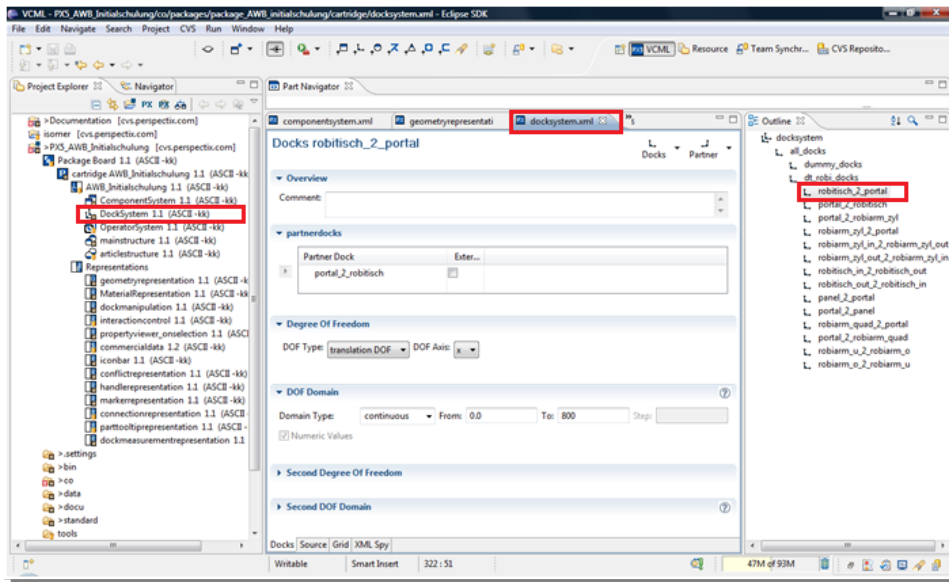
The **Weight** property now also has a slider.



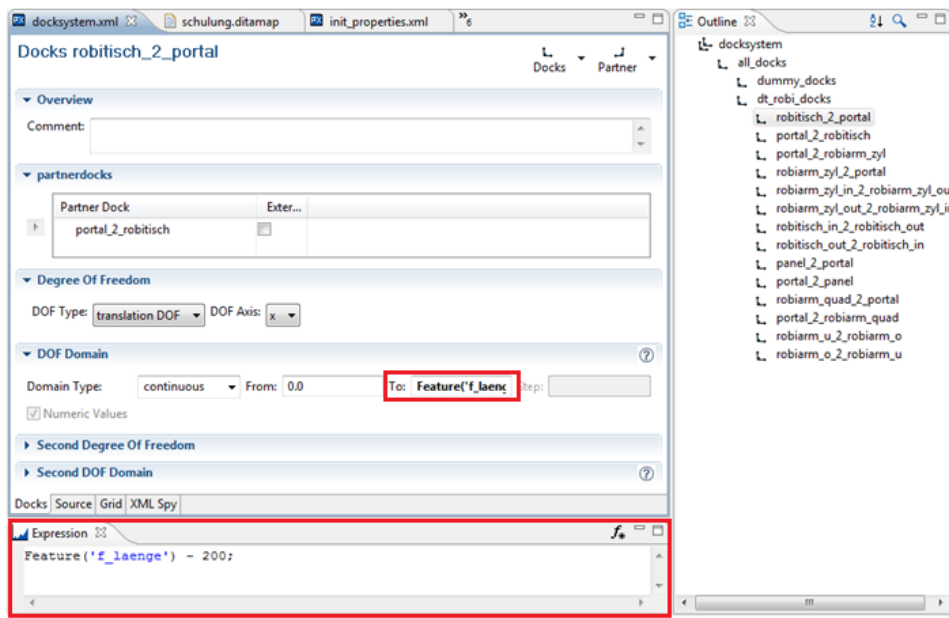
When you drag the gantry from the **Iconbar** into the **3D Scene** and connect it to the robot table, you will notice that the gantry can only be adjusted from 0 to 1000 units, even though you extended the table length to 1600 units. The reason is that the dock on the robot table with the degree of freedom in x-direction is still restricted from 0 to 1000 units.

38. For this dock, you must also enter the `Feature('f_length');` function for the translation in x-direction, like in the **Geometryrepresentation Editor**. To do this, open the **Docksystem**

Editor and select the `do_table_2_gantry` dock in the **Outline** under the `dt_robot_docks` entry.



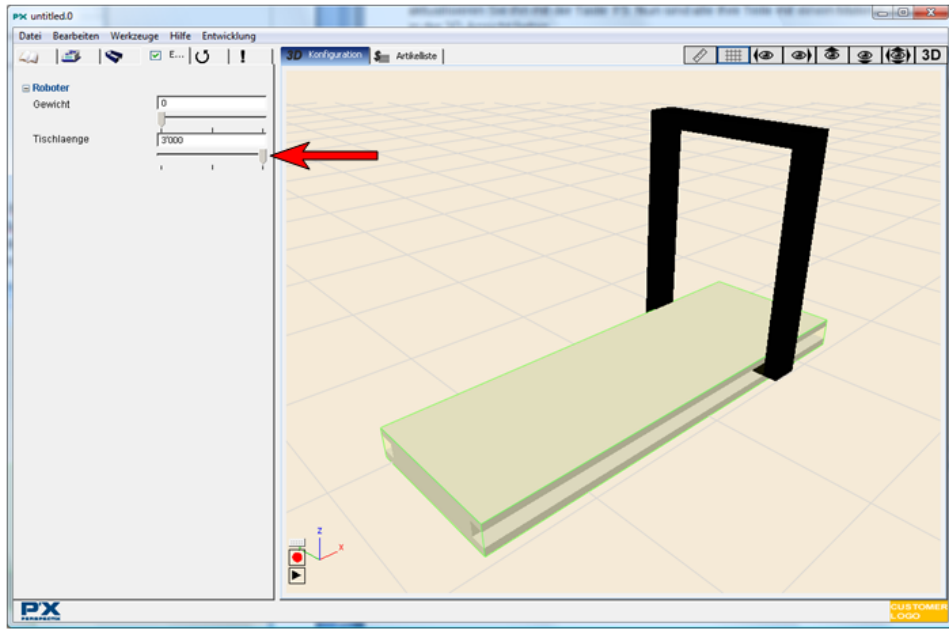
39. Replace 800 with the `Feature('f_length') - 200;` function in the **To** field in the **Dock-system Editor**. The 200 units must be subtracted from the possible total length, so that the gantry does not extend beyond the edges of the robot table (the width of the gantry is 200 units). In order to enter the function more easily, choose **Window > Show View > Expression** from the main menu to open the **Expression View**.
40. Save the **Docksystem Editor**.



Result

1. Start the **Configurator** or, if it is still open, press F5 to update it.
2. Drag the robot table from the **Iconbar** into the **3D Scene**.
3. Drag the gantry from the **Iconbar** onto the robot table.

4. Select the robot table in the **3D Scene** and click the **Properties** tab.
5. Extend the robot table with the **Table length** property and drag the gantry.



Now, you can create assemblies and rules.

Lesson 8 - Create Assemblies and Rules

You can assemble single components into assemblies.

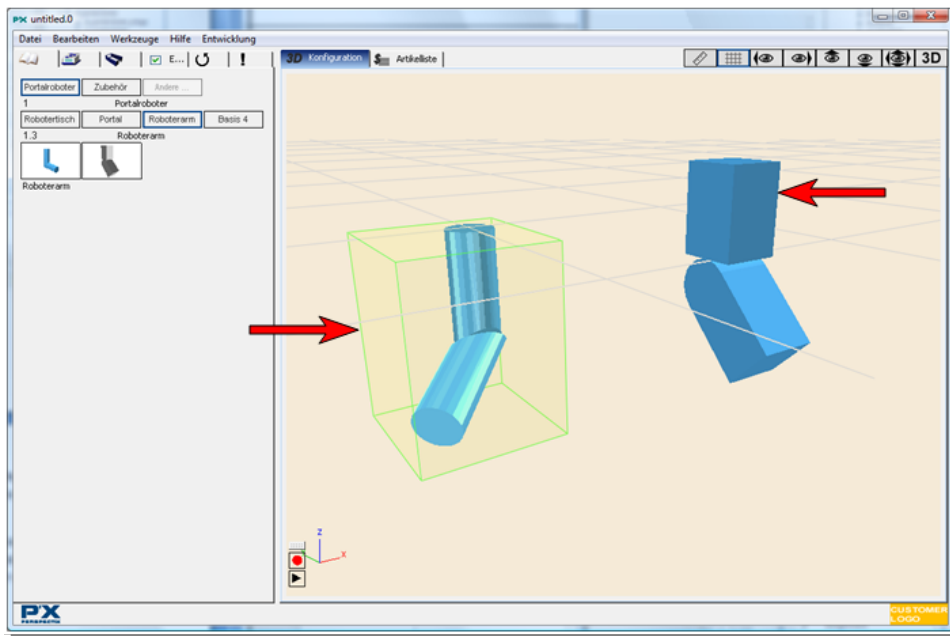
If you connect single components in the 3D Scene, the components will create an assembly. If you then click one of these components, the entire assembly will be selected. You can also depict the assembly as an icon in the Iconbar and then simply drag the assembly from the Iconbar into the 3D Scene.

Example

Drag both the round and square robot arms from the Iconbar into the 3D Scene.

When you click the round robot arms, both robot arms are selected. A rule defines that the robot arms form an assembly. To only select one round robot arm, double-click the robot arm.

When you click the square robot arms, only the robot arm that the mouse cursor is on is selected.



Assembly Types

There are two assembly types:

- Bottom-up assembly
- Top-down assembly

A bottom-up assembly is created when the matching components are connected in the 3D Scene. A top-down assembly is already assembled and can, e.g. be depicted as an icon in the Iconbar. An assembly can also be disassembled into its single components.

Your Assignment

Create a bottom-up assembly using the square robot arms:

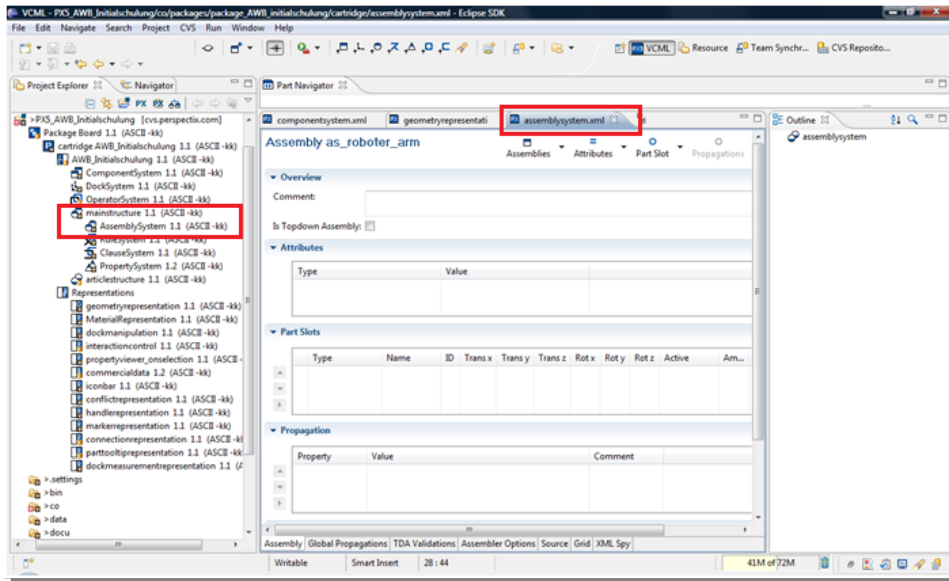
Assembly	Assembly Type
as robterarm quad	Bottom-up assembly

Assign the following rules to the assemblies:

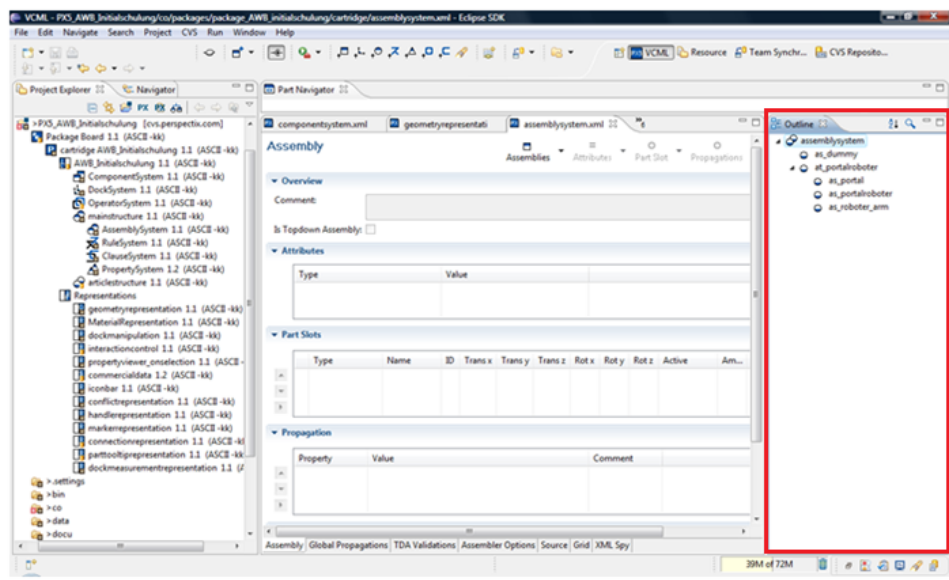
Rule	Referenced Assembly (Target Type)	Condition
asru_roboterarm_quad	as_roboterarm_quad	co_upper_robotarm_quad,co_lower_robotarm_quad
asru_portalroboter_var2	as_gantryroboter	as_gantry,as_roboterarm_quad,ct_control_panel

Procedure

1. Open the **Assemblysystem Editor**. The **Assembly System** is located in the **Project Explorer** under tutorial/mainstructure.



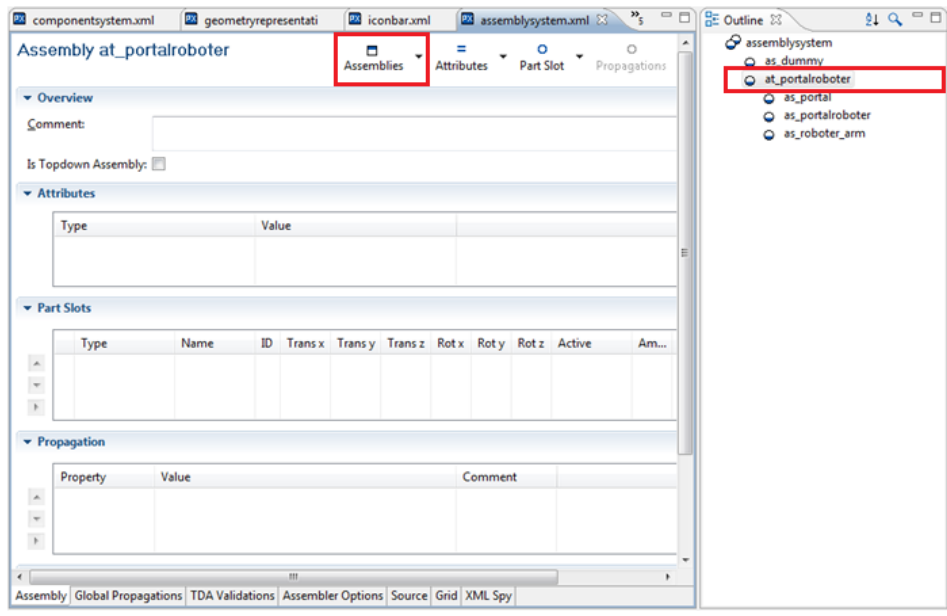
2. The **Outline** of the **Assemblysystem Editor** contains the already created assemblies. Click the arrow to the left of the entry to open the list. The arrow appears when you place the mouse cursor over the entry.



3. First, declare the assembly in the **Assemblysystem Editor**. Then create the rule in the **Assemblyrules Editor** that defines that the two square robot arms should form an assembly when they

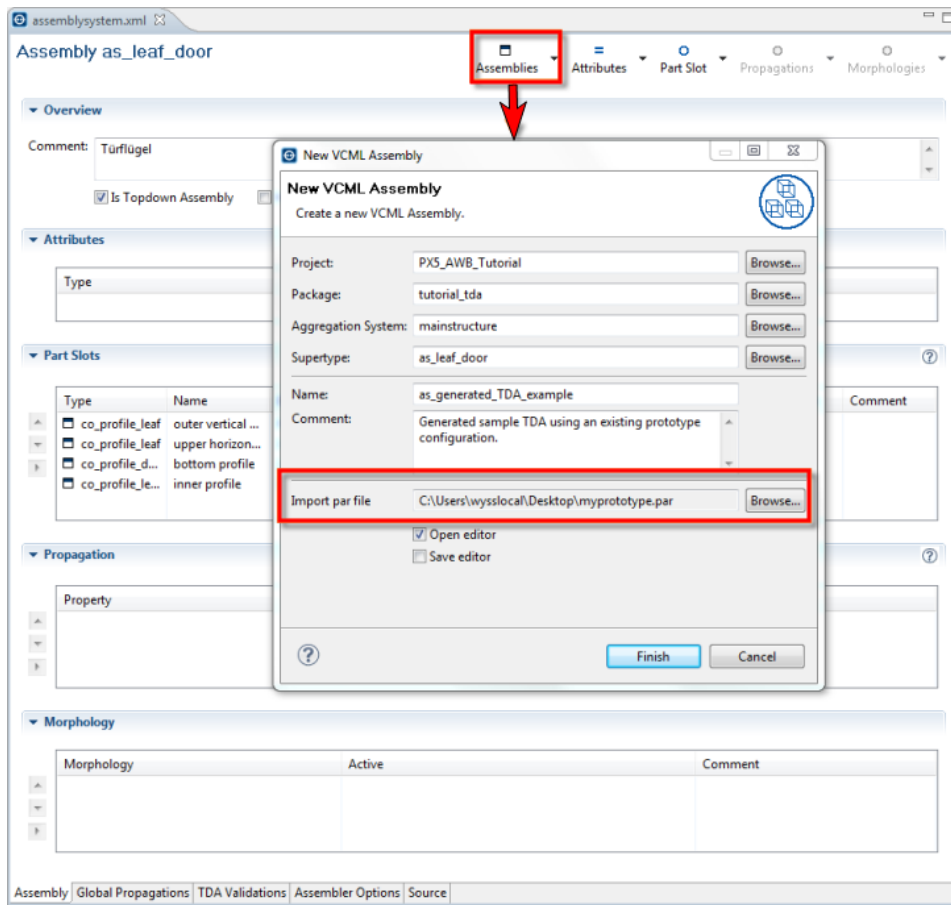
are connected in the **3D Scene**. The procedure is similar to creating components and geometries.

4. Select `at_portalroboter` in the **Outline** of the **Assemblysystem Editor** and click the **Assemblies** button.

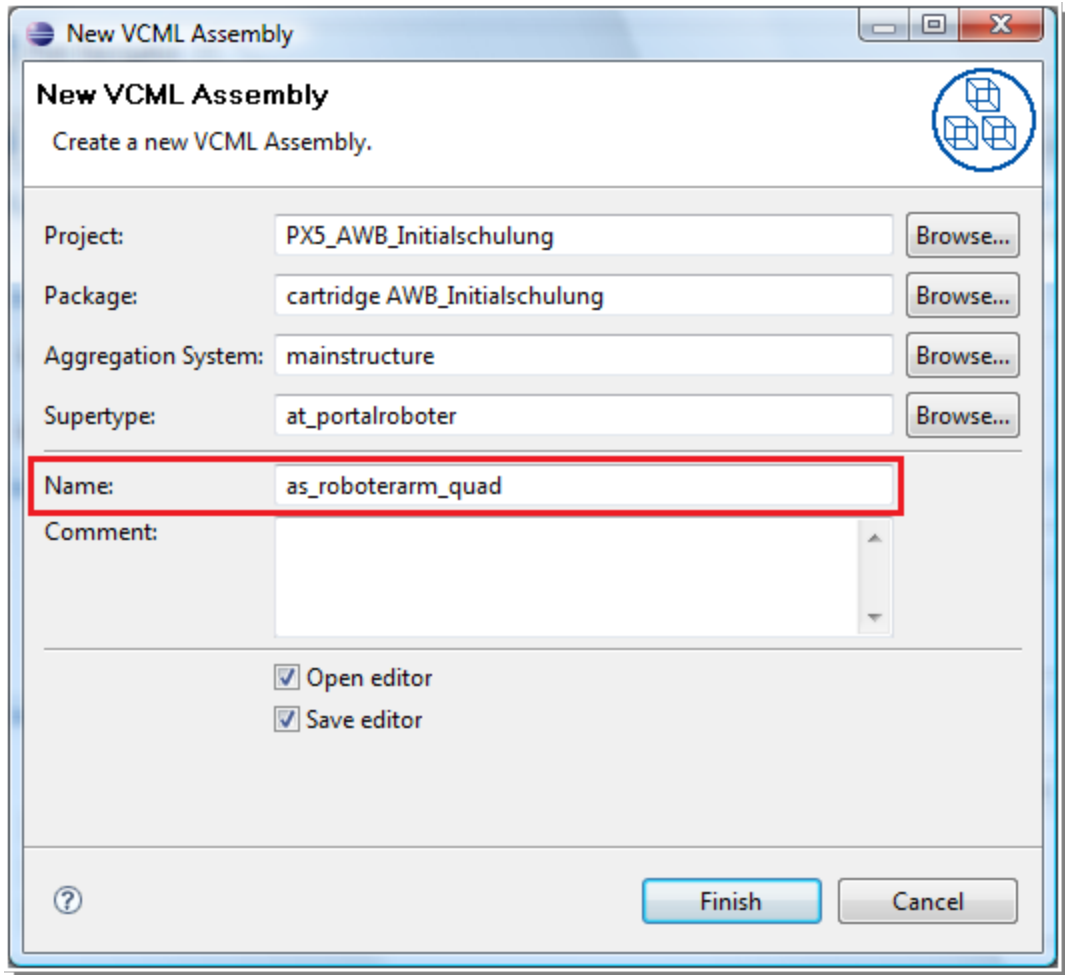


The **New VCML Assembly** window already contains a few entries. Your assembly will be placed hierarchically below the `at_portalroboter` entry in the **Supertype** field. Higher-ranking assembly types contain the `at_` prefix and lower-ranking assembly types contain the `as_` prefix. Higher-ranking assembly types are used for hierarchically ordering assemblies.

You can also import an assembly as a PAR file (`.par`). To do this, select the PAR file in the **Import par file** field in the **New VCML Assembly** window.

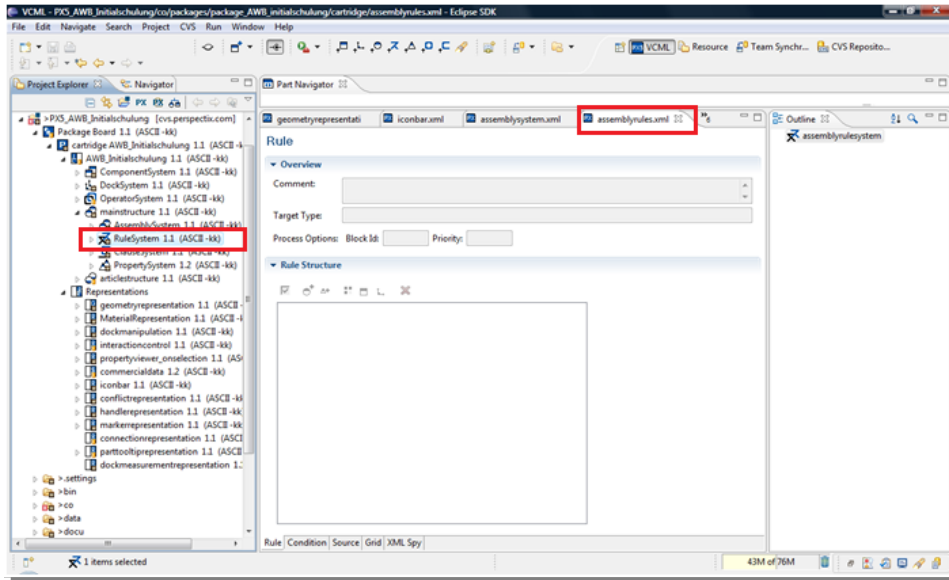


5. Type `as_roboterarm_quad` in the **Name** field in the **New VCML Assembly** window and click the **Finish** button.

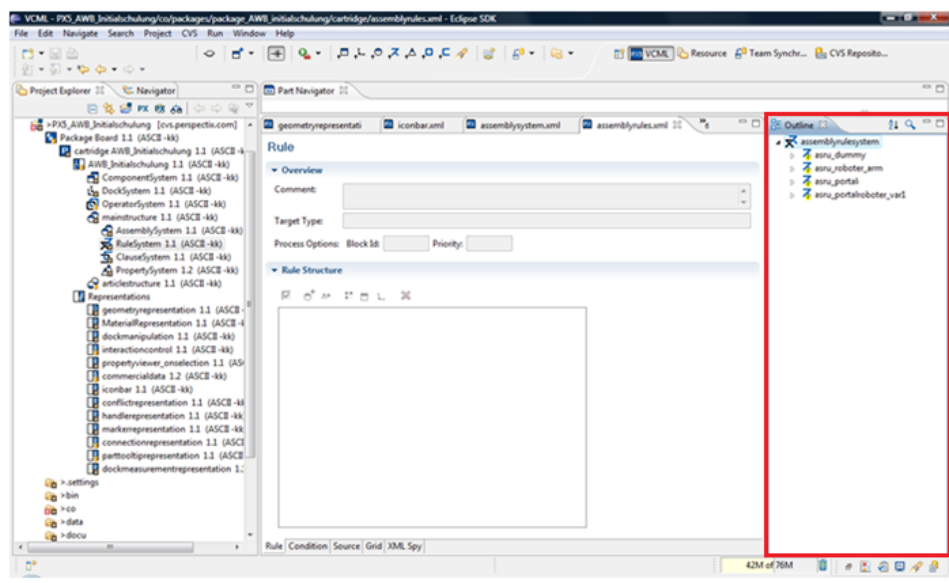


The **Outline** of the **Assemblysystem Editor** now contains the `as_roboterarm_quad` entry underneath the `at_portalroboter` entry.

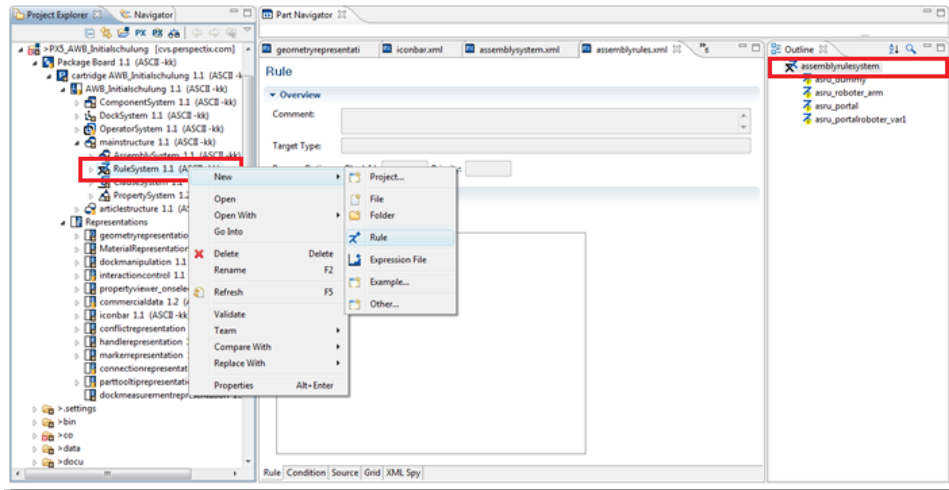
6. Save the **Assemblysystem Editor**.
7. Now you must create the rule, that under the `as_roboterarm_quad` assembly the two square robot arms should connect. To do this, open the **Assemblyrules Editor**. The **Rule System** is located in the **Project Explorer** under `tutorial/mainstructure`.



8. The **Outline** of the **Assemblyrules Editor** contains the already created rules. Click the arrow to the left of the entry to open the list. The arrow appears when you place the mouse cursor over the entry.

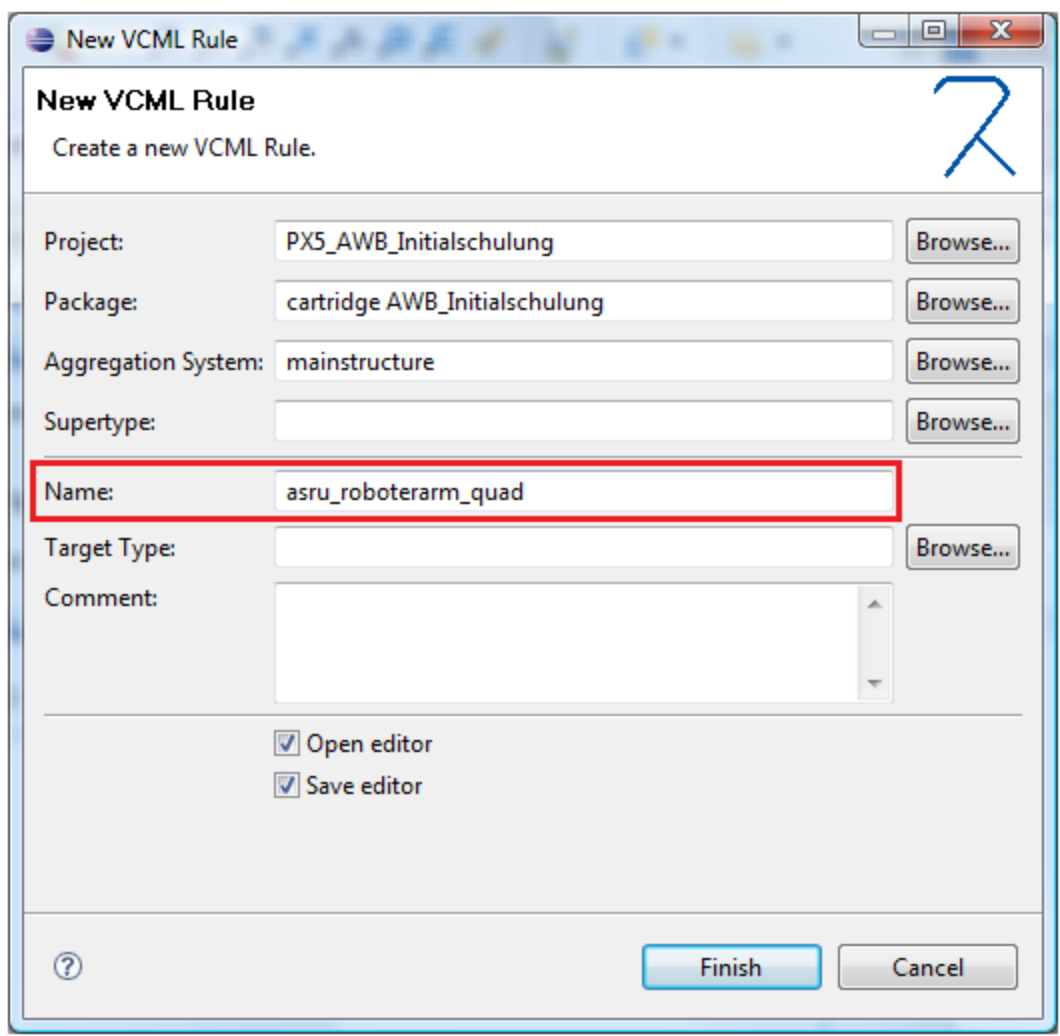


9. To create a rule, select the uppermost entry in the **Outline**.
10. Right-click **RuleSystem** in the **Project Explorer** and choose **New > Rule** from the context menu.



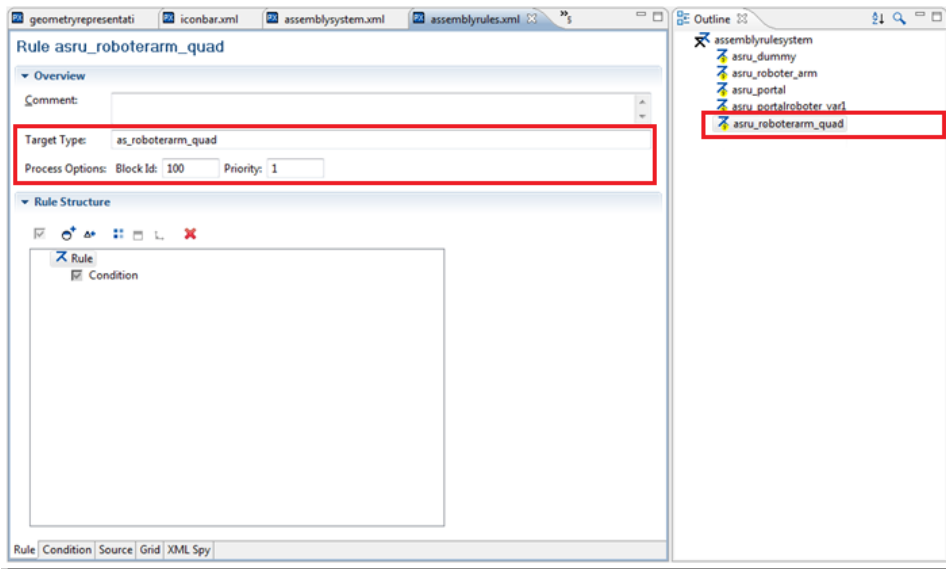
11. Type **asru_roboterarm_quad** in the **Name** field in the **New VCML Rule** window and click the **Finish** button.

Note: The name contains the **asru_** prefix and means "assembly rule". The prefix for assembly rules is required, so that assembly rules can easily be distinguished from other elements in the code.

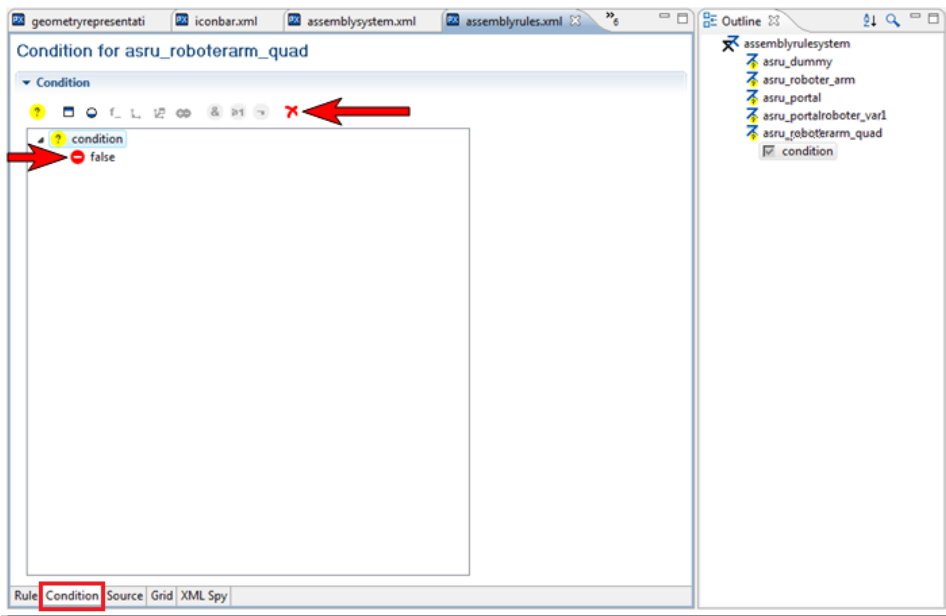


The **Outline** of the **Assemblyrules Editor** now contains the `asru_roboterarm_quad` entry.

12. Now you must create the rule in the **Assemblyrules Editor**. Type `as_roboterarm_quad` in the **Target Type** field or press CTRL+SPACEBAR and select the desired name from the pop-up window.
13. Type 100 in the **Block ID** field and 1 in the **Priority** field.



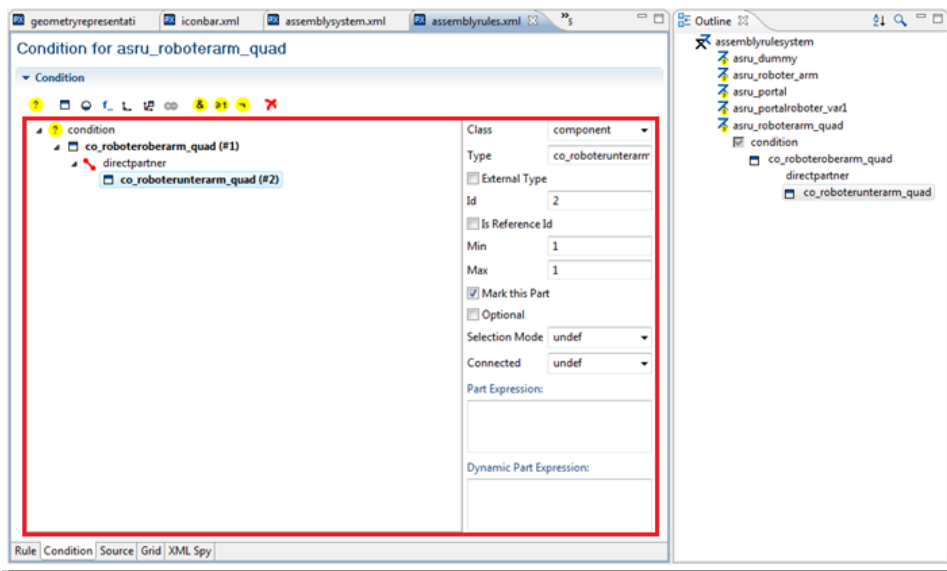
14. Open the **Condition** page in the **Assemblyrules Editor**. A condition, which you must delete, is already available by default.
15. Click the sign next to the `false` entry and then click the red cross in the iconbar above.



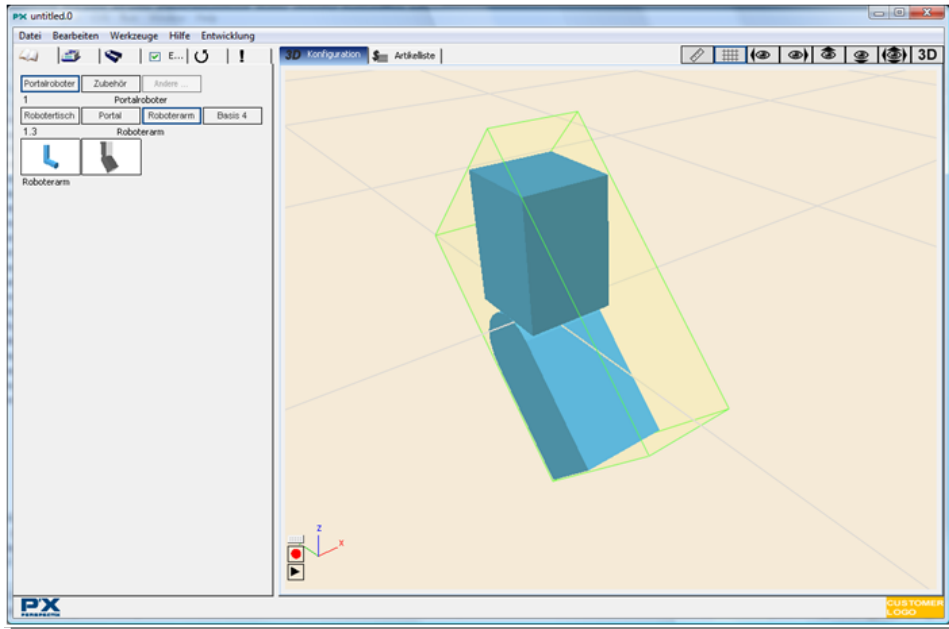
16. The rule for the assembly should define that the square robot upper arm and square robot lower arm should connect.
17. To do this, select `condition` in the middle pane.

Note: If the `condition` entry is not visible, click the yellow question mark on the left side in the iconbar.

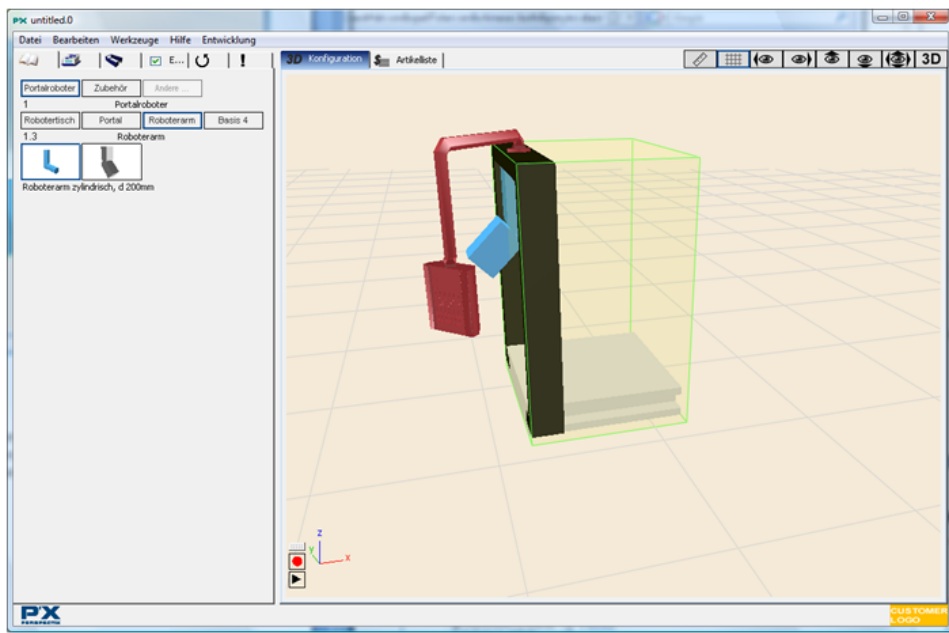
18. Click the second (squared) icon from the left in the iconbar, to add the first component that belongs to the condition.
19. Type `co_upper_robotarm_quad` in the **Type** field.
20. Type 1 in the **ID**, **Min** and **Max** fields. This component should appear only once in this assembly.
21. Select the **Mark this Part** checkbox.
22. Click the second (squared) icon from the left in the iconbar, to add the second component that belongs to the condition.
23. Type `co_lower_robotarm_quad` in the **Type** field.
24. Type 2 in the **ID** field and 1 in the **Min** and **Max** fields. This component should also appear only once in this assembly.
25. Select the **Mark this Part** checkbox.
26. Save the **Assemblyrules Editor**.



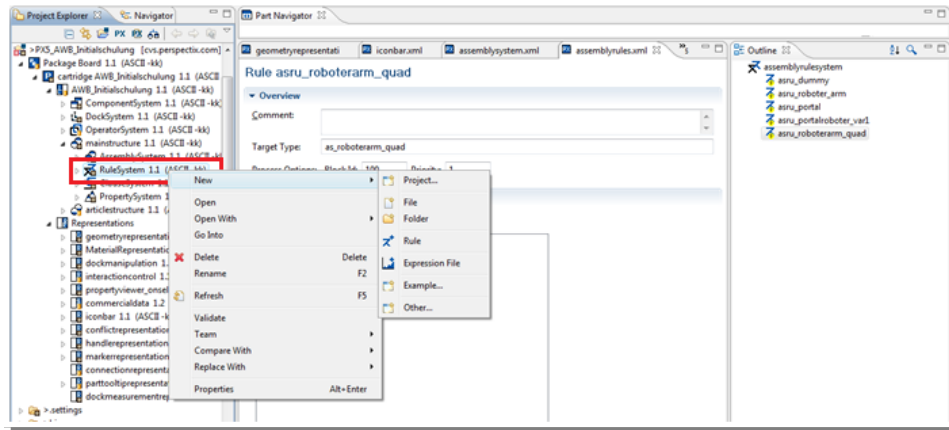
27. If the **Configurator** is still open it, close it and start it anew. All necessary data will be loaded again.
28. Drag the square robot arm from the **Iconbar** into the **3D Scene**. Now the entire robot arm will be selected, when you select a part. If you only want to select one part, to e.g. disassemble the assembly, double-click that part.



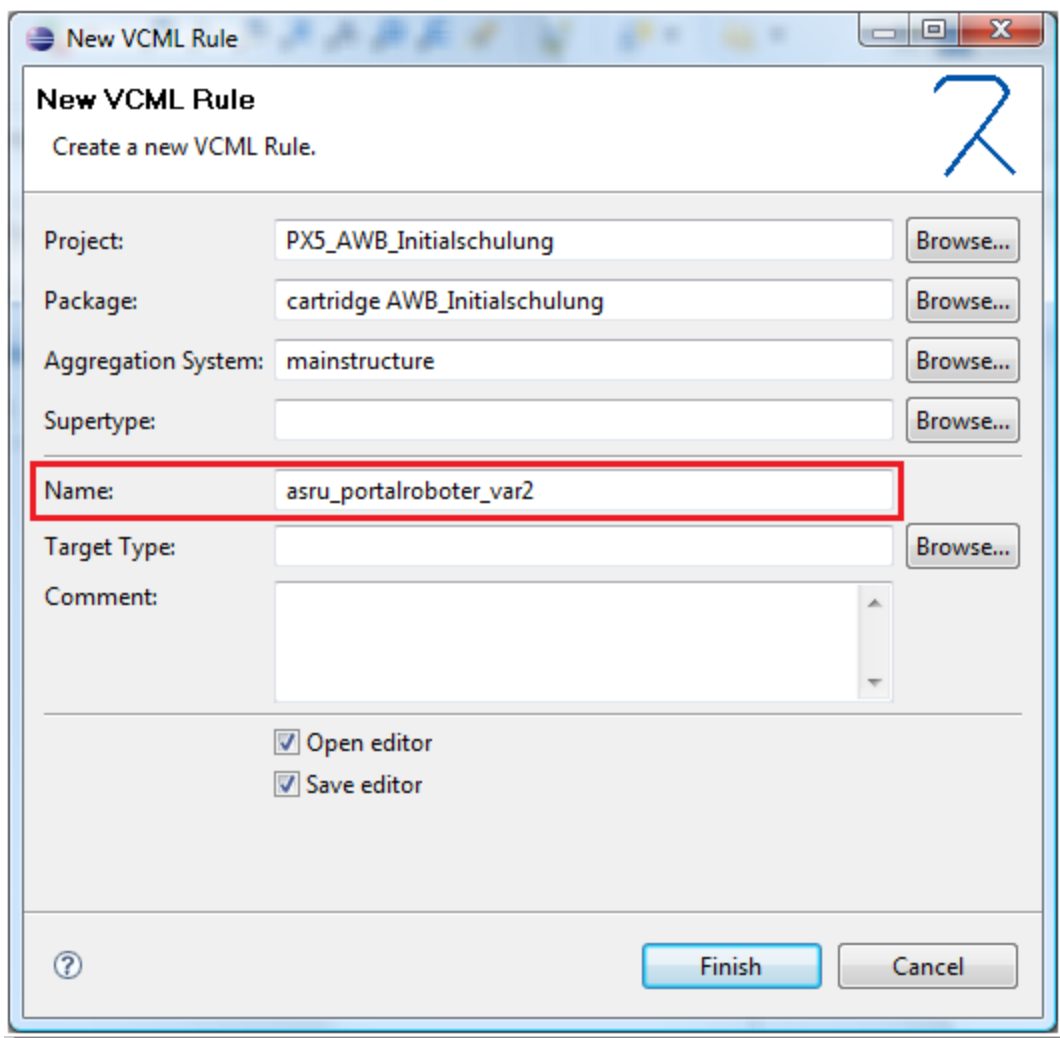
29. You will see that some components assemble into assemblies and some do not assemble. When you look at the other rules in the **Rule System**, you will see that only those parts connect for which a rule exists.
30. Assemble the robot in the **Configurator** using the robot table, gantry, square robot arm, and control panel parts. You will see that not all parts can be assembled to an assembly. The new parts that you created, square robot arm and control panel do not belong to the assembly yet.



31. Open the **Assemblyrules Editor** and create a new rule, which assembles the robot table, gantry, and your newly created parts, square robot arm and control panel to an assembly.
32. Right-click **RuleSystem** in the **Project Explorer** and choose **New > Rule** from the context menu.



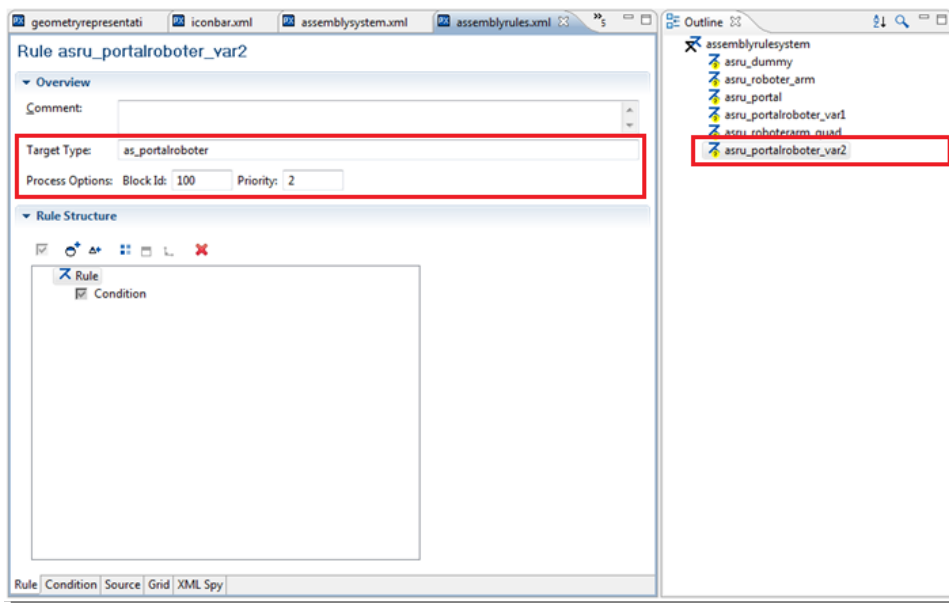
33. Type **asru_gantry_robot_var2** in the **Name** field in the **New VCML Rule** window and click the **Finish** button.



The **Outline** of the **Assemblyrules Editor** now contains the **asru_gantry_robot_var2** entry.

34. Now create the rule in the **Assemblyrules Editor**. To do this, type **as_gantryroboter** in the **Target Type** field or press CTRL+SPACEBAR and select a name from the pop-up window.

35. Type **100** in the **Block ID** field and **2** in the **Priority** field.

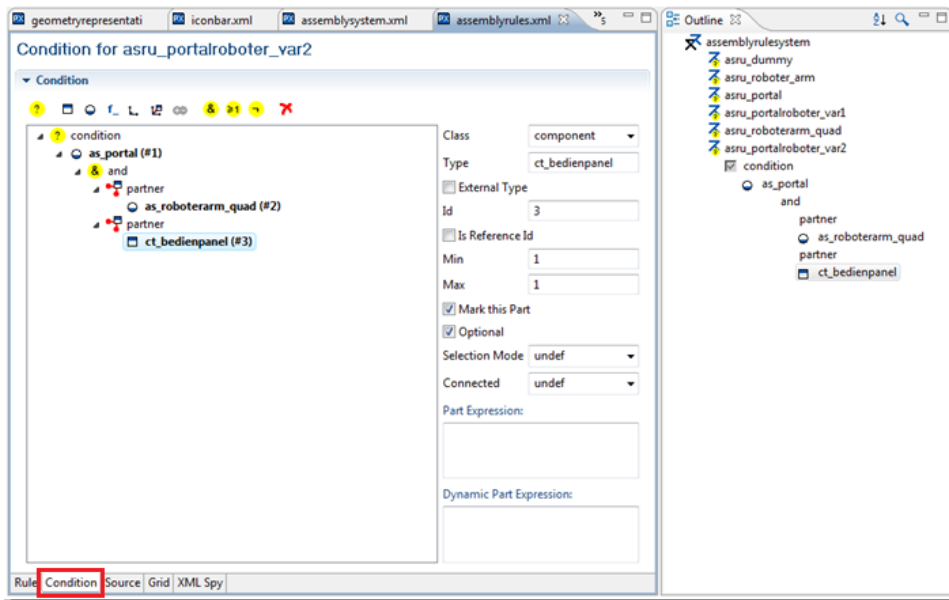


36. Open the **Condition** page in the **Assemblyrules Editor**. A condition, which you must delete, is already available by default.
37. Click the sign next to the `false` entry and then click the red cross in the iconbar above.
38. The last rule that you should create, defines that your parts, square robot arm and control panel should form an assembly with the robot table and gantry. To do this, select `condition` in the middle pane.

Note: If the `condition` entry is not visible, click the yellow question mark on the left side in the iconbar.

39. Click the third (round) icon from the left in the iconbar, to add an assembly that will belong to the condition.
40. Type `as_gantry` in the **Type** field.
41. Type **1** in the **ID**, **Min** and **Max** fields. This assembly should appear only once in this assembly.
42. Select the **Mark this Part** checkbox.
- You have now added an existing assembly to this assembly, compared to your previous rule, where you only added components.
43. Click the third (round) icon from the left in the iconbar, to add a second assembly that will belong to the condition.
44. Type `as_roboterarm_quad` in the **Type** field.
45. Type **2** in the **ID** field and **1** in the **Min** and **Max** fields. This assembly should also appear only once in this assembly.
46. Select the **Mark this Part** checkbox.
47. These two assemblies must be connected as partners. To do this, select the entry between the two assemblies and select `partner` in the **Class** list box.

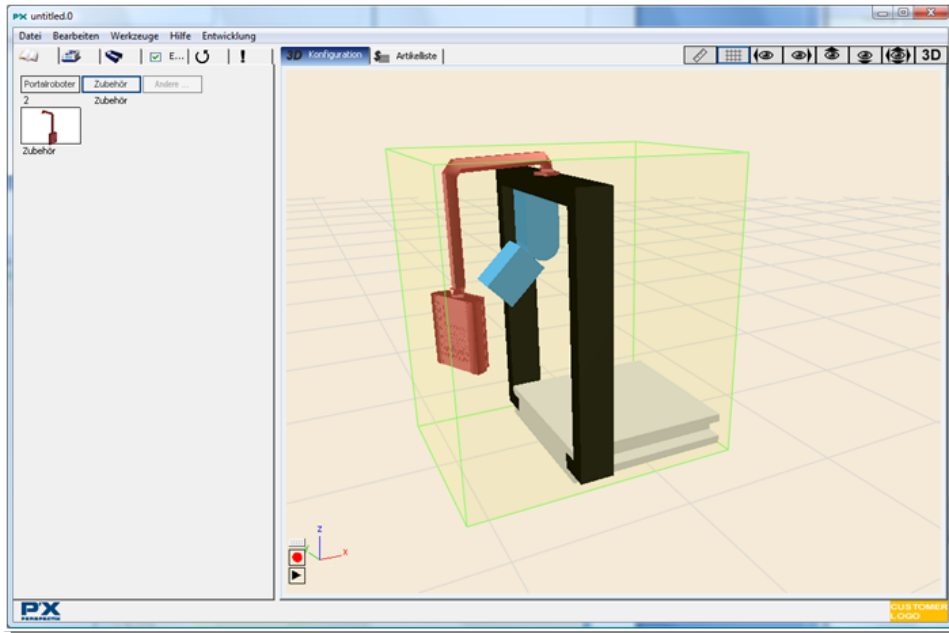
48. The control panel should also belong to the assembly. Select the `as_gantry` assembly in the **Condition** field and click the second (squared) icon from the left in the iconbar, to add a component.
49. Type `ct_control_panel` in the **Type** field.
50. Type 3 in the **ID** field and 1 in the **Min** and **Max** fields.
51. Select the **Mark this Part** checkbox.
52. The `ct_control_panel` component must also be connected with the `as_gantry` assembly as a partner. To do this, select the entry above the `ct_control_panel` entry and select `partner` in the **Class** list box.
53. Save the **Assemblyrules Editor**.



Result

1. If the **Configurator** is still open, close it and start it anew. All necessary data will be loaded again.
2. Assemble the robot using the robot table, gantry and your new parts, the square robot arm and the control panel.

The parts should connect and form an assembly.



Now, you can create an article list.

Lesson 9 - Create Article List

An article list needs to be created for the parts that are assembled in the Configurator. The article list can either contain single parts or entire assemblies. You can also create separate assemblies for the article list, which are only meant for the article list.

The advantage is that when an article is only sold with additional articles, you can define an assembly, which does not need to be displayed in the 3D Scene, e.g. a part is only delivered with specific screws that are too small to be displayed in the 3D Scene.

In an article list parts can be listed:

- As a single part, like it is shown in the 3D View.
- As an assembly, like it is defined in the assembly system. For the bottom-up assembly the advantage is that the article is created when the corresponding components are assembled to an assembly in the 3D View.
- As an assembly, like it is defined in the article system. The advantage is that additional parts for an existing part or assembly can be listed in the article list, without having to show them in the 3D View.

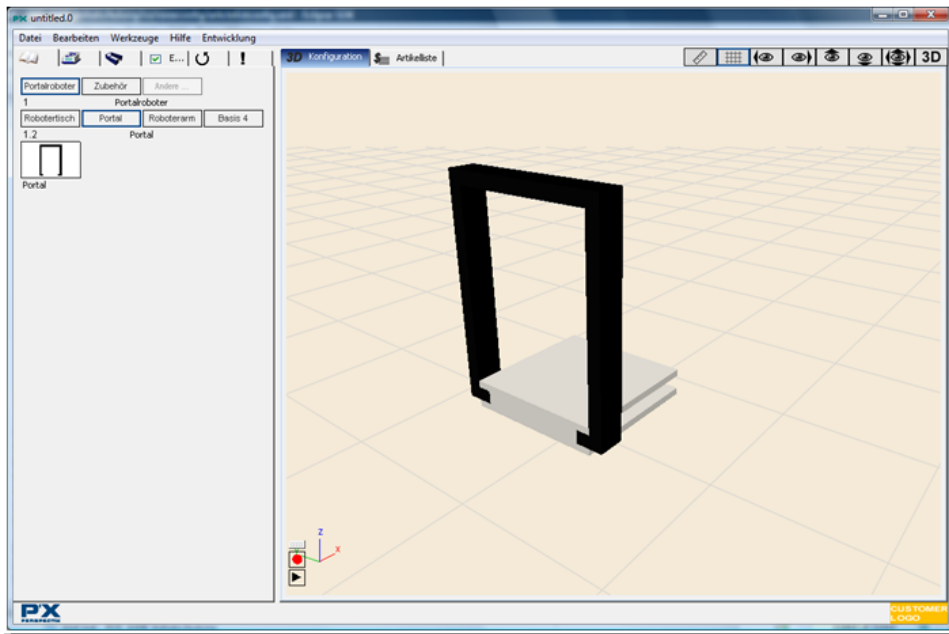
In our example an article list already exists for the robot.

Your Assignment

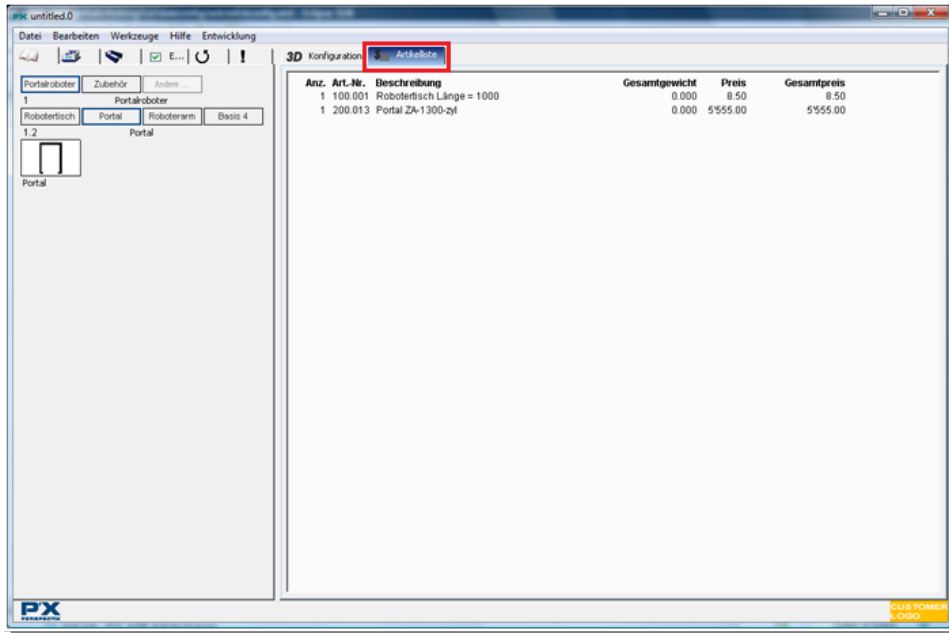
Define articles, which are listed in the article list, for your newly created components (parts) and assemblies.

Procedure

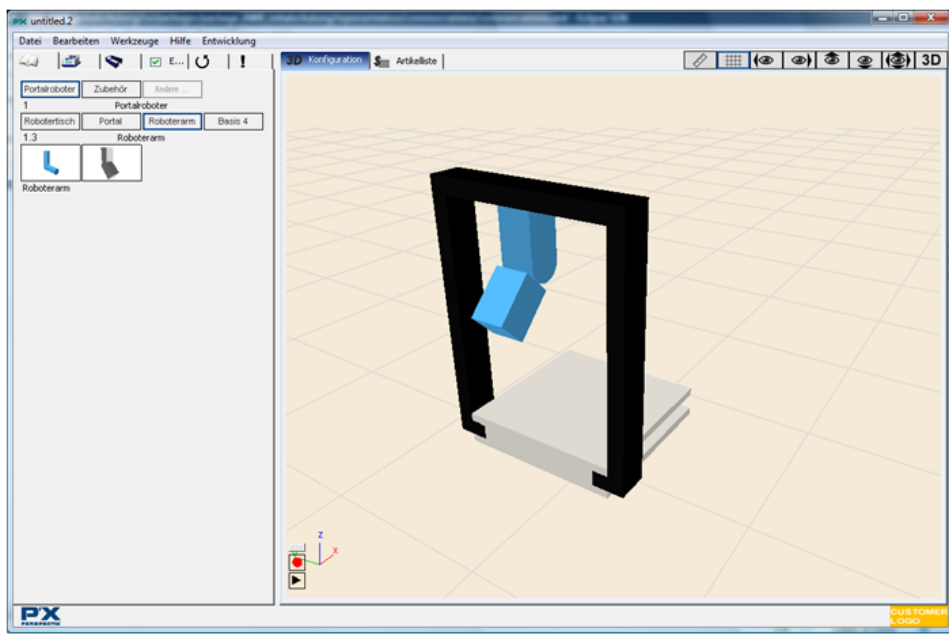
1. Open the **Configurator** and drag the robot table and a gantry from the **Iconbar** into the **3D View**.



2. Open the **Article List**. The parts that are visible in the **3D View** are listed in the article list.

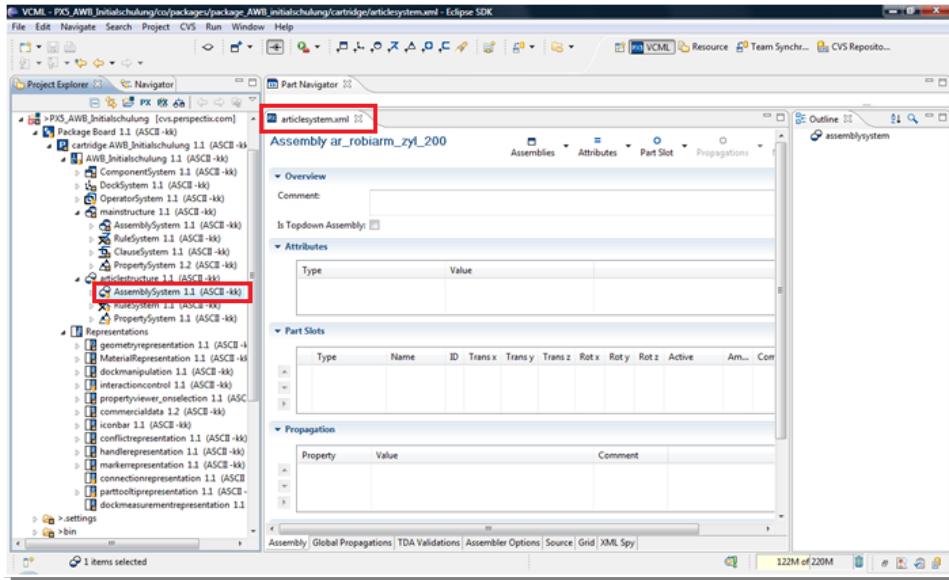


3. Drag the square robot arm from the **Iconbar** onto the robot in the **3D View**.

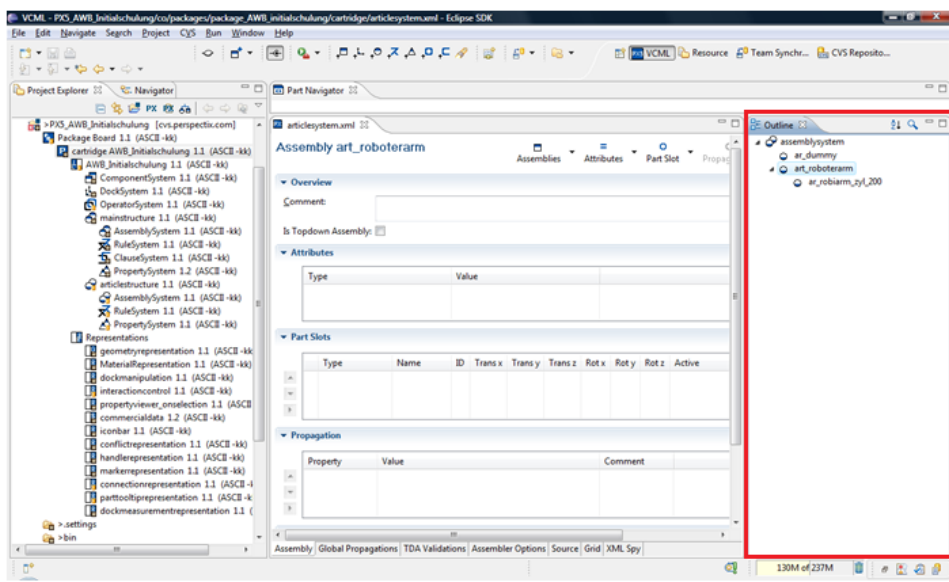


4. Open the **Article List**. You will see that your newly created components are not listed in the **Article List**. You must first add your newly created components to the article structure.
5. Open the **Assembly System** in the **Project Explorer** under **articlestructure**.

Note: You have already worked with the **Assembly System** under **mainstructure**. This is the **Assembly System** under the **articlestructure**. The difference is that in the main structure you can define assemblies, which are shown in the 3D View, and in the article structure you can define assemblies, which are listed in the article list.

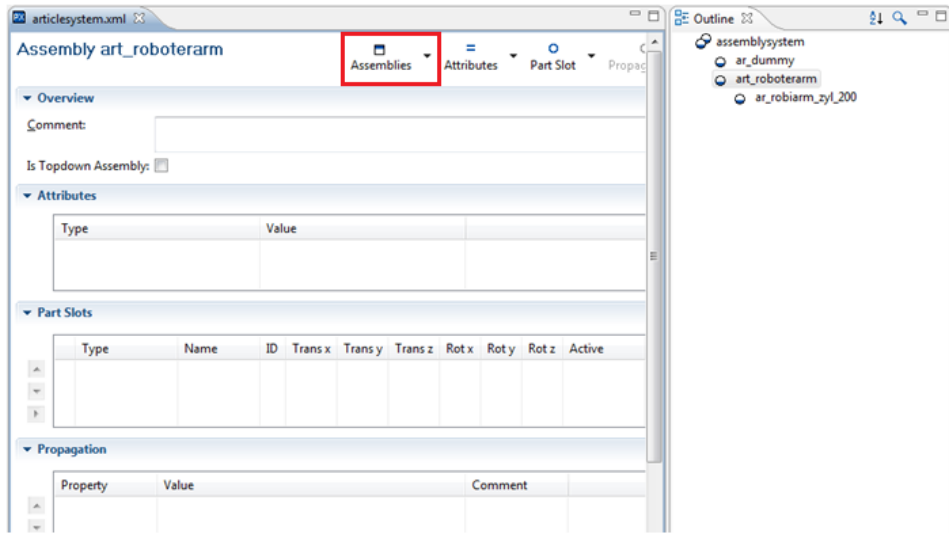


- Take a look at the already created assemblies (article groups) in the **Outline** of the **Articlesystem Editor**. Click the arrow to the left of the `assemblysystem` entry. The arrow appears when you place the mouse cursor over the entry.



Just like in the main structure, you must create a bottom-up assembly here using the two square robot arms. First, declare the assembly in the **Articlesystem Editor**. Then, create the rule in the **Rule System** of the **Article System** that defines that the two square robot arms should form an assembly, which is listed in the **Article List**.

- Select `art_roboterarm` in the **Outline** and click the **Assemblies** button.

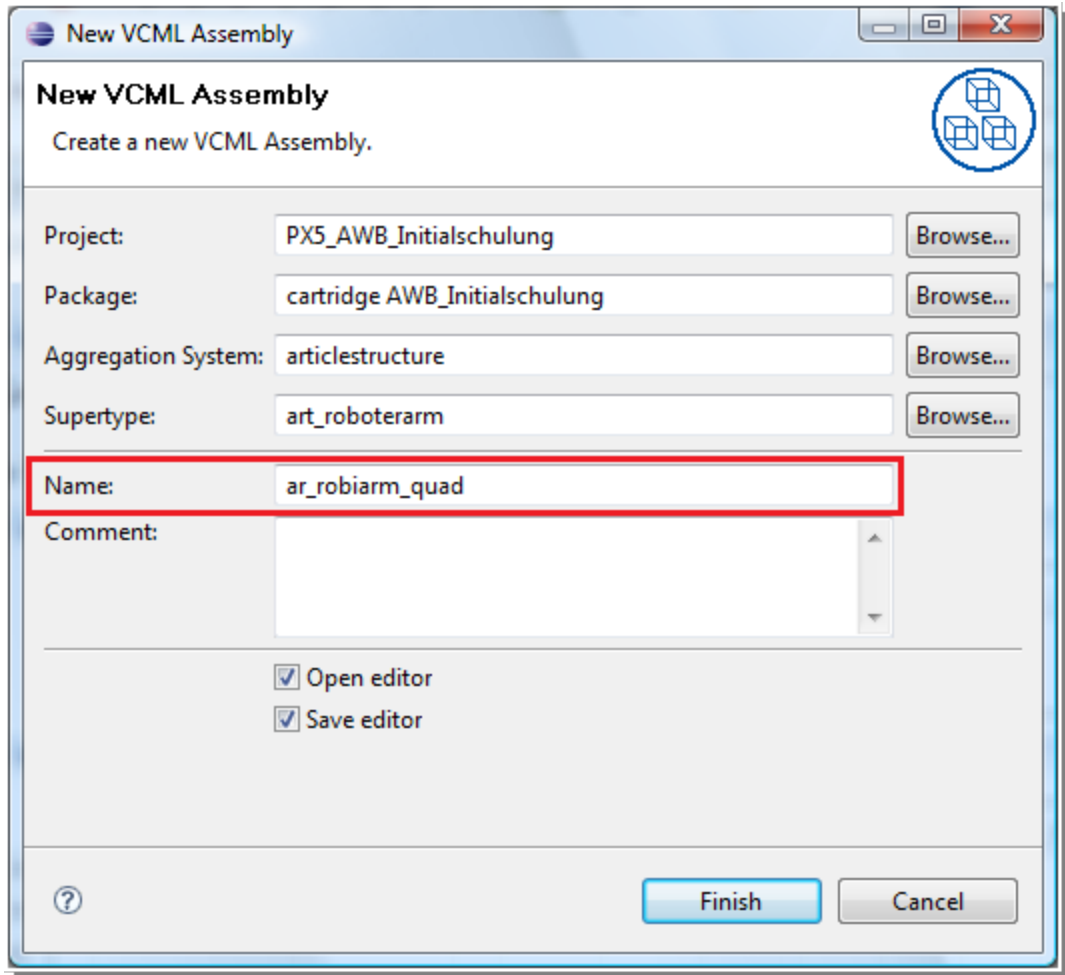


The **New VCML Assembly** window already contains a few entries.

Your assembly will be placed hierarchically below the `art_roboterarm` entry in the **Super-type** field. In the article structure, higher-ranking assembly types contain the `art_` prefix and lower-ranking assembly types contain the `ar_` prefix. Higher-ranking assembly types are used for hierarchically ordering assemblies.

Note: Note the difference to the **Assembly System** of the main structure, where higher-ranking types contain the `at_` prefix and lower-ranking types contain the `as_` prefix.

8. Type `ar_robarm_quad` in the **Name** field in the **New VCML Assembly** window and click the **Finish** button.

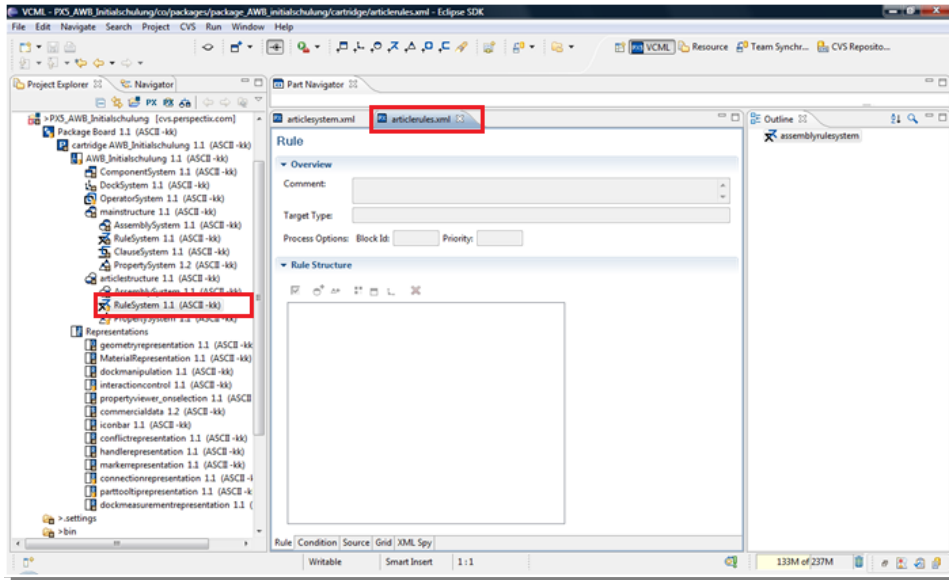


The **Outline** of the **Articlesystem Editor** now contains the `ar_robiarm_quad` entry below the `art_roboterarm` entry.

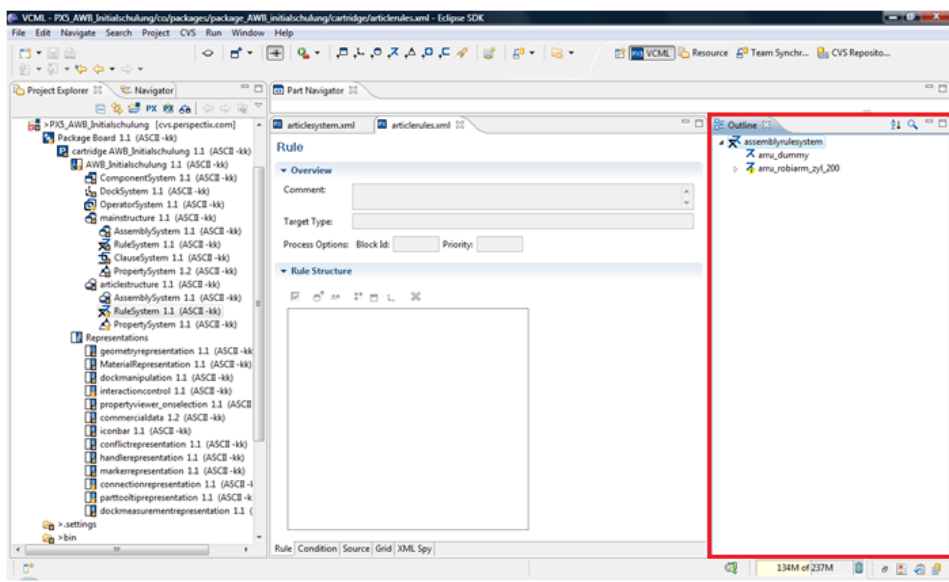
9. Save the **Articlesystem Editor**.

Now you must create the rule that defines that the square robot arms will be listed as articles under the `ar_robiarm_quad` assembly.

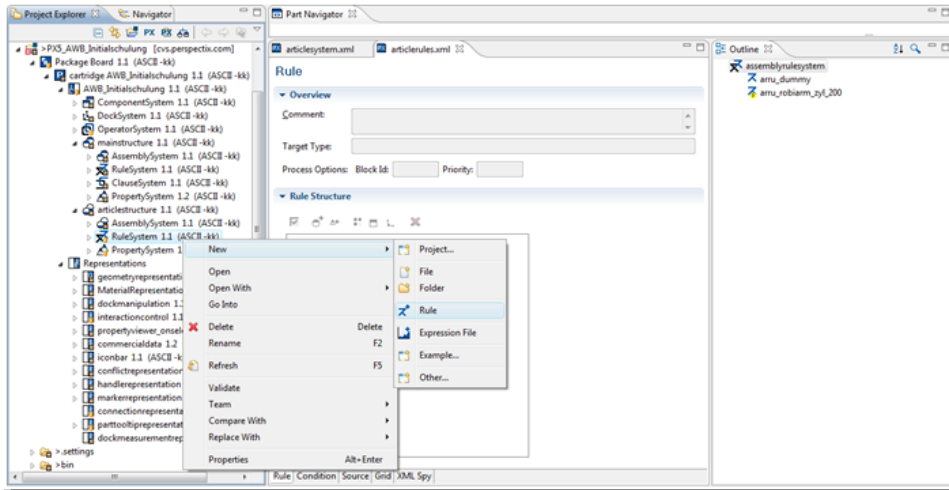
10. Open the **Articlerules Editor** in the article structure. The **Rule System** is located under `AWB_Initialschulung/articlestructure`.



11. Take a look at the existing rules in the article structure in the **Outline** of the **Articlerules Editor**. Click the arrow to the left of the `assemblyrulesystem` entry. The arrow appears when you place the mouse cursor over the entry.

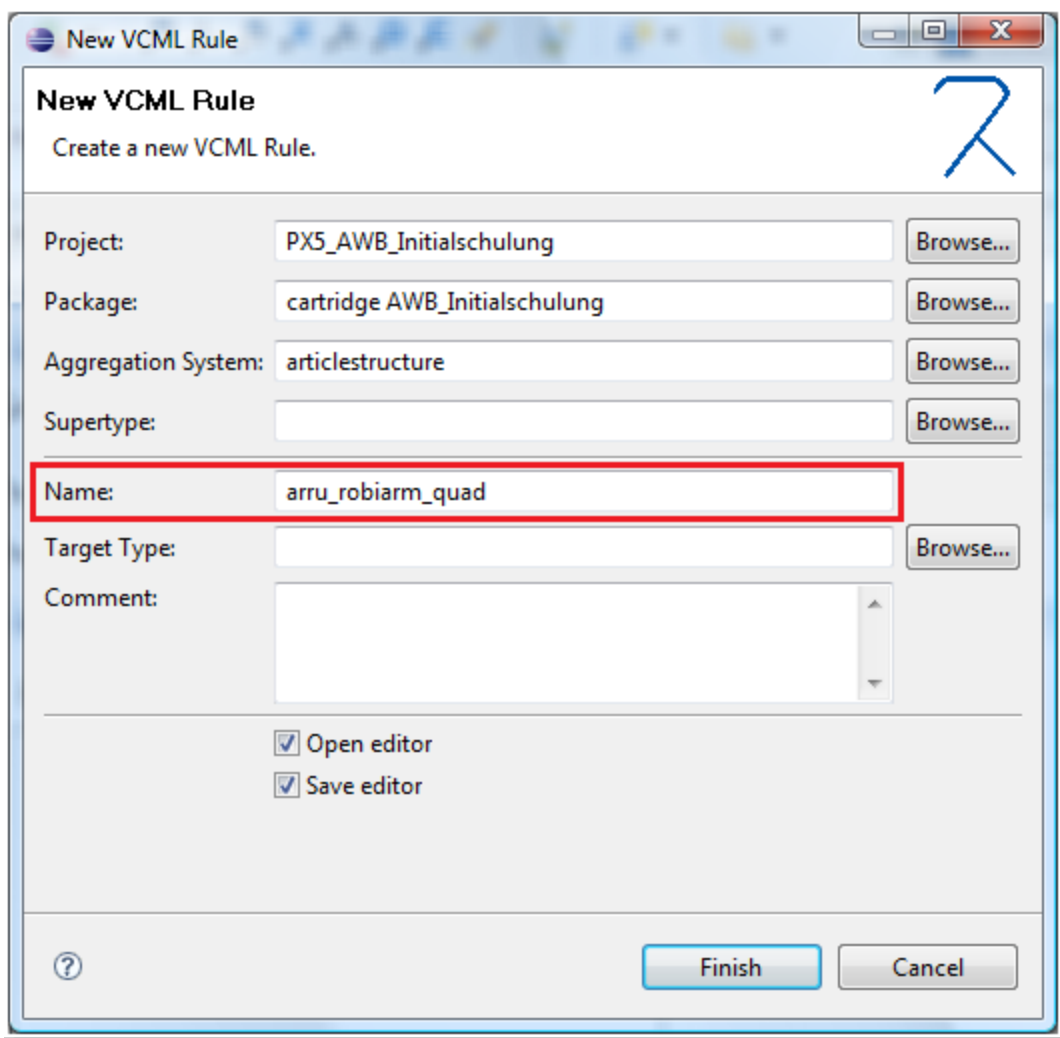


12. Now create the rule. To do this, select the uppermost entry in the **Outline**.
13. Right-click **Rule System** in the article structure in the **Project Explorer** and choose **New > Rule** from the context menu.



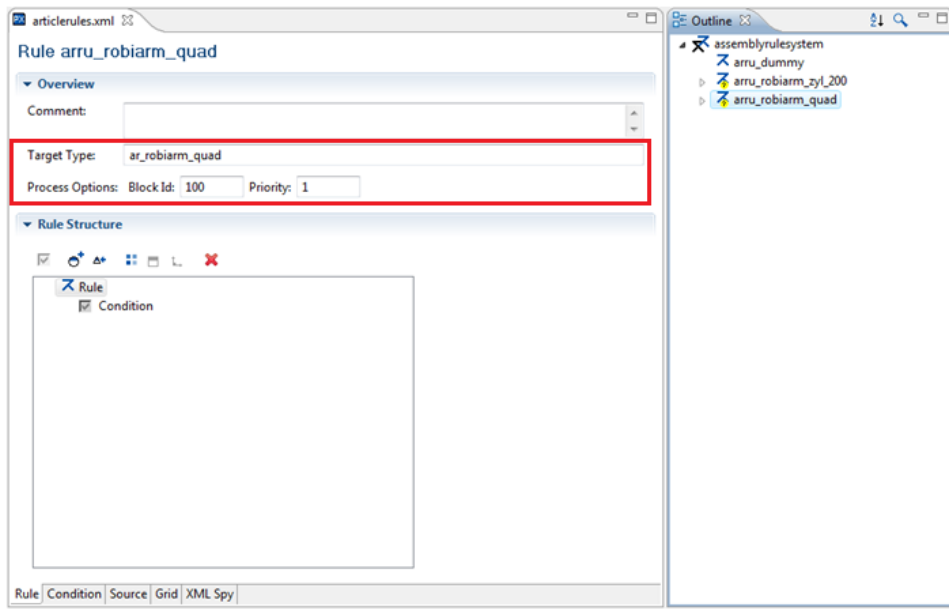
14. Type **arru_robiarm_quad** in the **Name** field in the **New VCML Rule** window and click the **Finish** button.

Note: The name contains the **arru_** prefix and means "article rule". The prefix for article rules is required, so that article rules can easily be distinguished from other elements in the code.

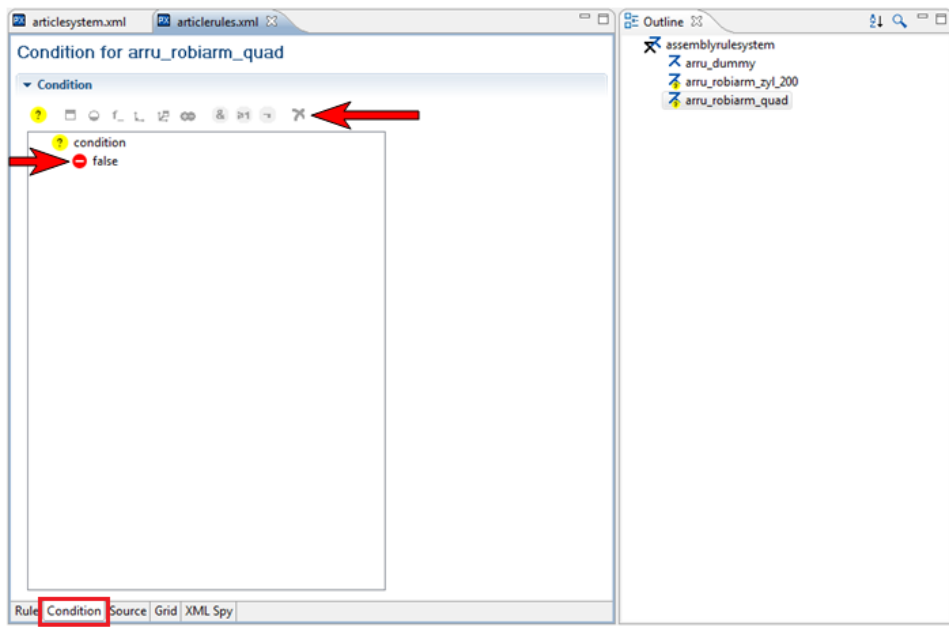


The **Outline** of the **Articlerules Editor** now contains the `arsru_robiarm_quad` entry. Now you must create the rule in the **Articlerules Editor**.

15. Type `ar_robiarm_quad` in the **Target Type** field or press CTRL+SPACEBAR and select an entry from the pop-up window.
16. Type 100 in the **Block ID** field and 1 in the **Priority** field.
17. Save the **Articlerules Editor**.



18. Open the **Condition** page in the **Articlerules Editor**. A condition, which you must delete, is already available by default.
19. Click the sign next to the `false` entry and then click the red cross in the iconbar above.

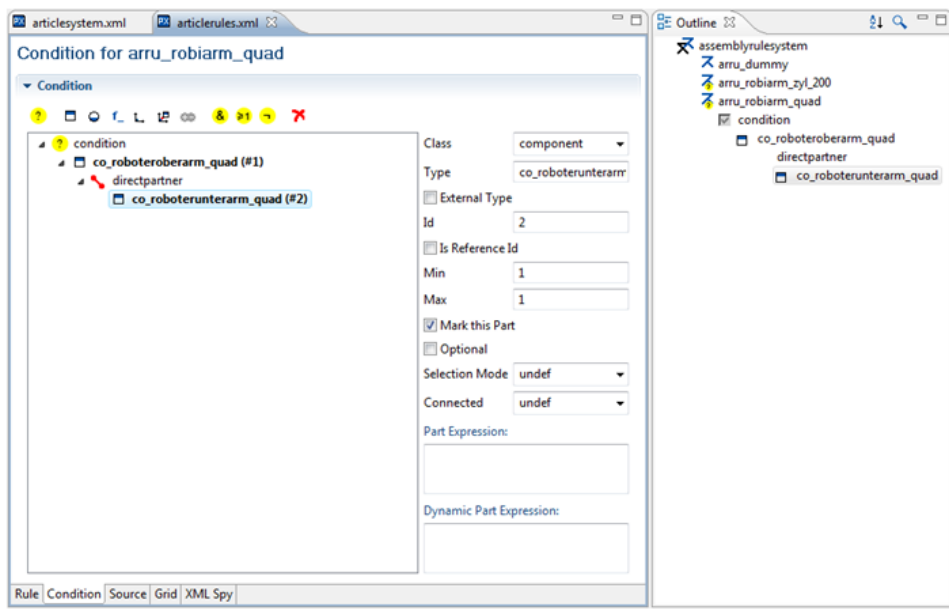


20. The rule for the square robot arm article, which should be listed in the **Article List**, should define that it is made up of the square robot upper arm and the square robot lower arm. To do this, select

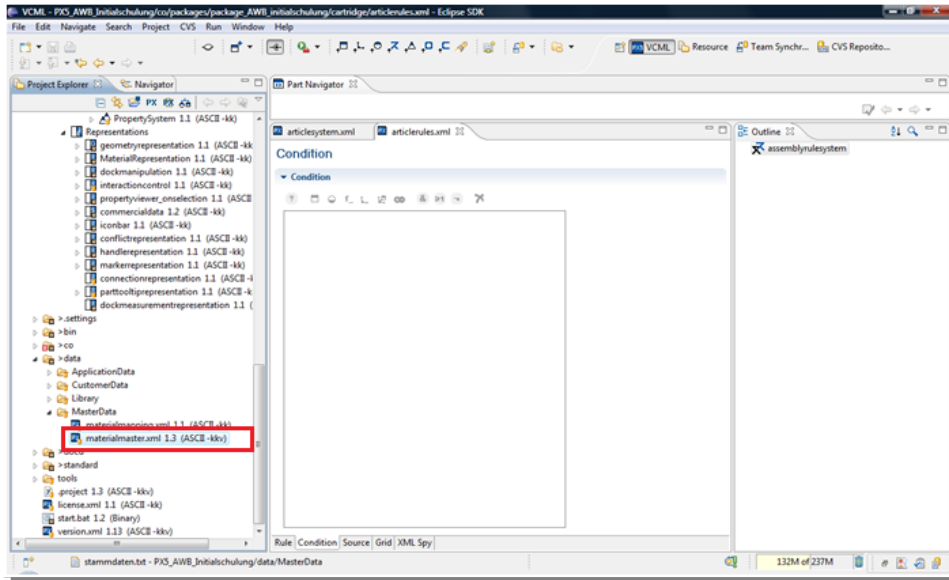
condition in the middle pane.

Note: If the `condition` entry is not visible, click the yellow question mark on the left side in the iconbar.

21. Click the second (squared) icon from the left in the iconbar, to add the first component that belongs to the condition.
22. Type `co_upper_robotarm_quad` in the **Type** field.
23. Type 1 in the **ID**, **Min** and **Max** fields. This component should appear only once in this assembly.
24. Select the **Mark this Part** checkbox.
25. Click the second (squared) icon from the left in the iconbar, to add the second component that belongs to the condition.
26. Type `co_lower_robotarm_quad` in the **Type** field.
27. Type 2 in the **ID** field and 1 in the **Min** and **Max** fields. This component should also appear only once in this assembly.
28. Select the **Mark this Part** checkbox.
29. Save the **Articlerules Editor**.



30. In order for the individual parts to be listed in the **Article List** in the **Configurator** exactly as they are defined by you, you must declare the definitions in the master data of the `materialmaster.xml` file. To do this, open the `materialmaster.xml` file using Excel. The file is located in the **Project Explorer** under `data/MasterData`.



The file has already been prepared for this example. At the beginning of this lesson, you have seen that the robot table and the gantry are listed in the **Article List**, but the square robot arm is not. The reason is that your components in the first column have not yet been assigned.

A	B	C	D	E	F	G
px_type	artikelnummer	preis_ch	bezeichnung_de	bezeichnung_en	angebotstext_de	angebotstext_en
co_robotertisch_var	100.001	8.5	Robotertisch Länge = 1000	Robitable length = 1000		
co_portal_zs_1300	200.013	5555	Portal ZA-1300-zyl	Portal ZA-1300-zyl		
	200.015		Portal ZA-1500-zyl	Portal ZA-1500-zyl		
	250.013		Portal ZA-1300-quad	Portal ZA-1300-quad		
	250.015		Portal ZA-1500-quad	Portal ZA-1500-quad		
co_roboterarm_zyl_200	300.013	466	Roboterarm KG-1300-zyl	Roboterarm KG-1300-zyl		
co_roboterunterarm_zyl_200	300.015	455	Roboterarm KG-1500-zyl	Roboterarm KG-1500-zyl		
	350.013		Portal KG-1300-quad	Portal KG-1300-quad		
	350.015		Portal KG-1500-quad	Portal KG-1500-quad		
	400.001	4444	Roboterarm zylindrisch D200	Robiarm zyl. D200		
	500.001	3333	Roboterarm quadratisch	Robiarm cube		
	999.099	2222	Bedienpanel Z99	Userpanel Z99		

You can see the already created entries in the third column, e.g. round robot arm (Roboterarm zylindrisch D200), square robot arm (Roboterarm quadratisch), and control panel (Bedienpanel Z99).

31. Type **ar_robiarm_quad** in the first column of the row with the **Roboterarm quadratisch** entry. You have defined this article group in the **Article System**.
32. Type **co_control_panel_99** in the first column of the row with the **Bedienpanel Z99** entry. This is the component for the control panel, which you defined in the **Component System**.

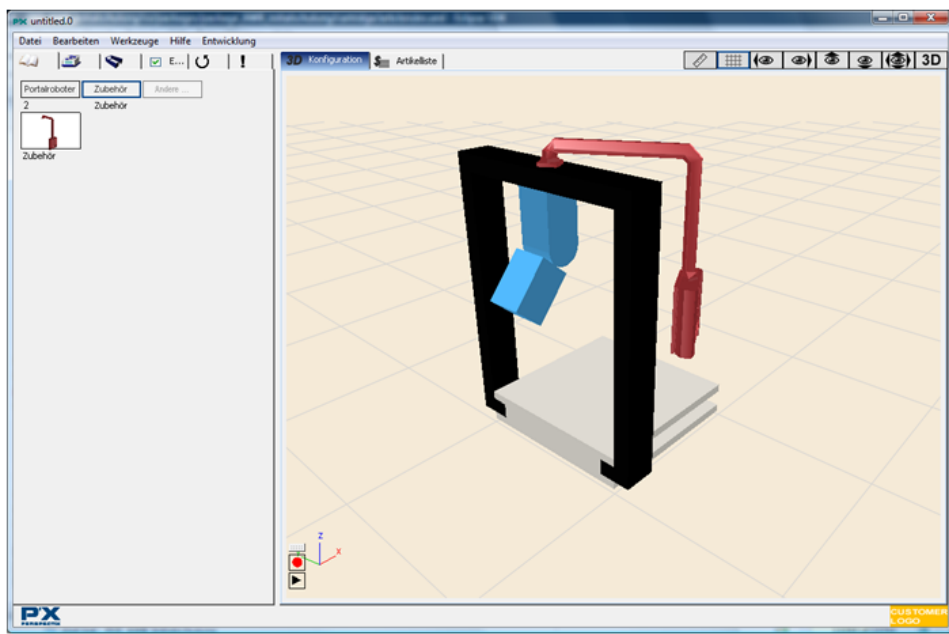
Note: Components for which no article has been defined in the **Article System** can also be listed in the `materialmaster.xml` file.

px_type	artikelnummer	preis_ch	bezeichnung_de	bezeichnung_en	angebotstext_de	angebotstext_en
co_robotertisch_var	100.001	8,5	Robotertisch Länge = 1000	Robitable length = 1000		
co_portal_zs_1300	200.013	5555	Portal ZA-1300-zyl	Portal ZA-1300-zyl		
	200.015		Portal ZA-1500-zyl	Portal ZA-1500-zyl		
	250.013		Portal ZA-1300-quad	Portal ZA-1300-quad		
	250.015		Portal ZA-1500-quad	Portal ZA-1500-quad		
co_roboterarm_zyl_200	300.013	466	Roboterarm KG-1300-zyl	Roboterarm KG-1300-zyl		
co_roboterarm_zyl_200	300.015	455	Roboterarm KG-1500-zyl	Roboterarm KG-1500-zyl		
	350.013		Portal KG-1300-quad	Portal KG-1300-quad		
	350.015		Portal KG-1500-quad	Portal KG-1500-quad		
	400.001	4444	Roboterarm zylindrisch D200	Robiarm zyl. D200		
ar_robotarm_quad	500.001	3333	Roboterarm quadratisch	Robiarm cube		
co_bedienspanel_z99	999.099	2222	Bedienpanel Z99	Userpanel Z99		

33. Save the `materialmaster.xml` file.

Result

1. If the **Configurator** is still open, close it and start it anew. All necessary data will be loaded again.
2. Assemble the robot using the robot table, gantry and your new parts, the square robot arm and the control panel Z99.

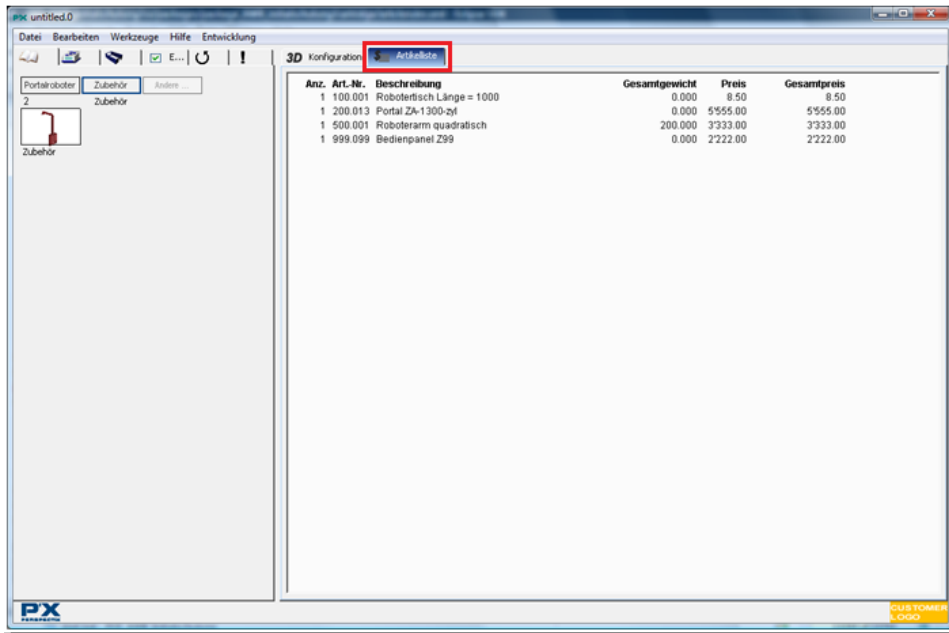


3. Open the **Article List**. The components you created are listed in the **Article List**.

Did you notice that the square robot arms are not listed individually, but as one article?

When you disassemble the robot arm in the **3D Scene**, it is no longer listed in the **Article List**. Try it out. Double-click the lower part of the robot arm, until it is selected and drag it away. Then,

switch to the **Article List** and you will see that the robot arm is no longer listed. Put the robot arm back together and take a look at the **Article List** again.



*Did you notice that another article group `ar_robarm_zyl_200` is listed in the **Article System**?*

The round robot arms are assigned to this article group. Type the `ar_robarm_zyl_200` article group in the first column of the row with the **Roboterarm zylindrisch D200** entry and delete the `co_roboteroberarm_zyl_200` and `co_roboterunterarm_zyl_200` entries in the `materialmaster.xml` file.

You will notice that the round robot arms are also not listed individually, but as an article group in the **Article List**. The article group is only listed if the round robot arms are connected.

Now, you can insert properties into the article list.

Lesson 10 - Insert Properties into Article List

You can also insert properties of components into the article list to calculate data.

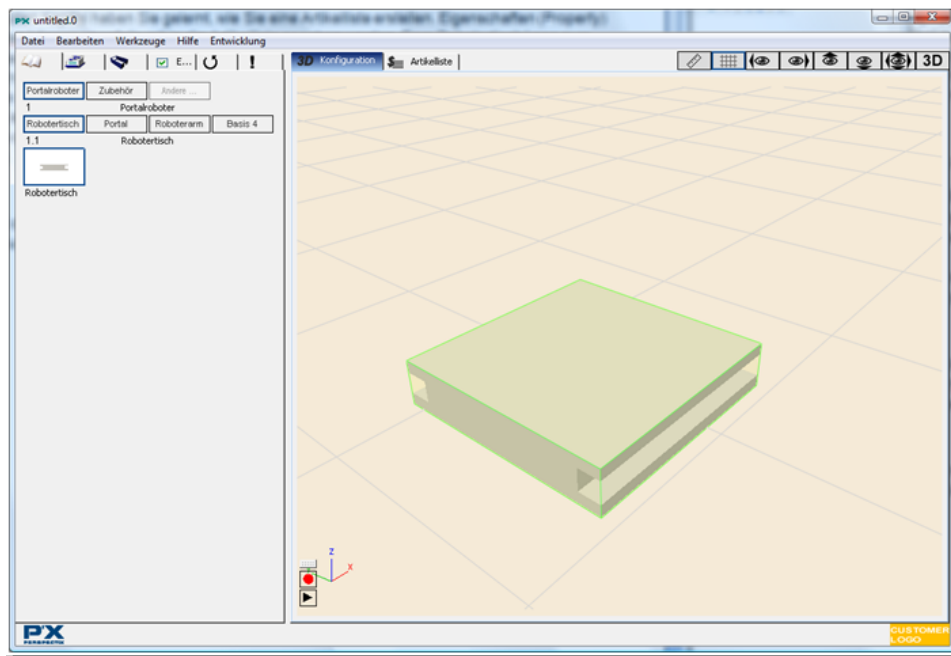
For example, if you change the length of a robot table, the new length is listed in the article list and the price of the robot table automatically adjusts to the length.

Your Assignment

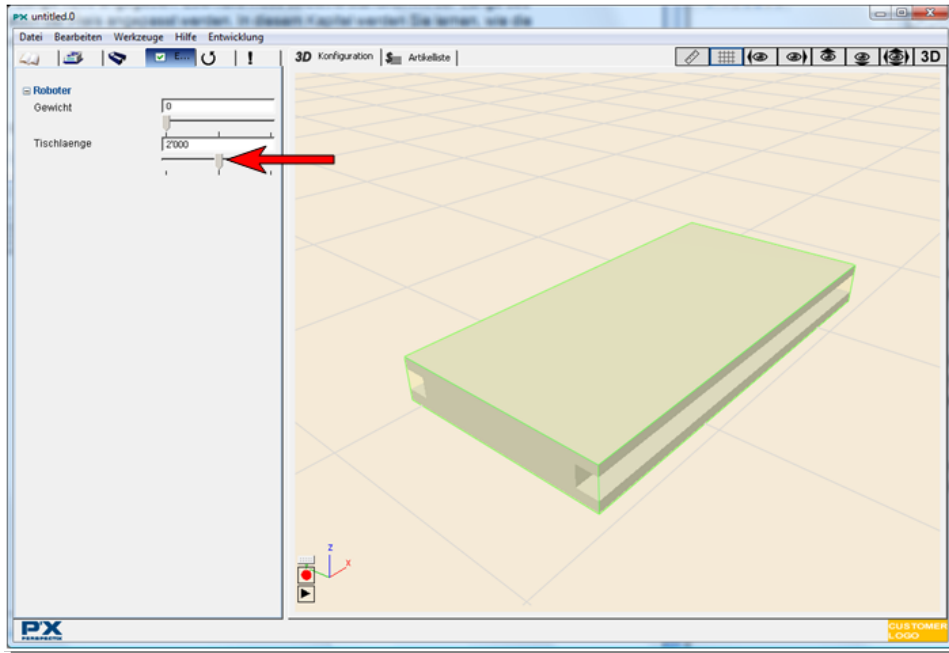
Insert the table length property of the robot table into the article list so that the price adjusts to the table length.

Procedure

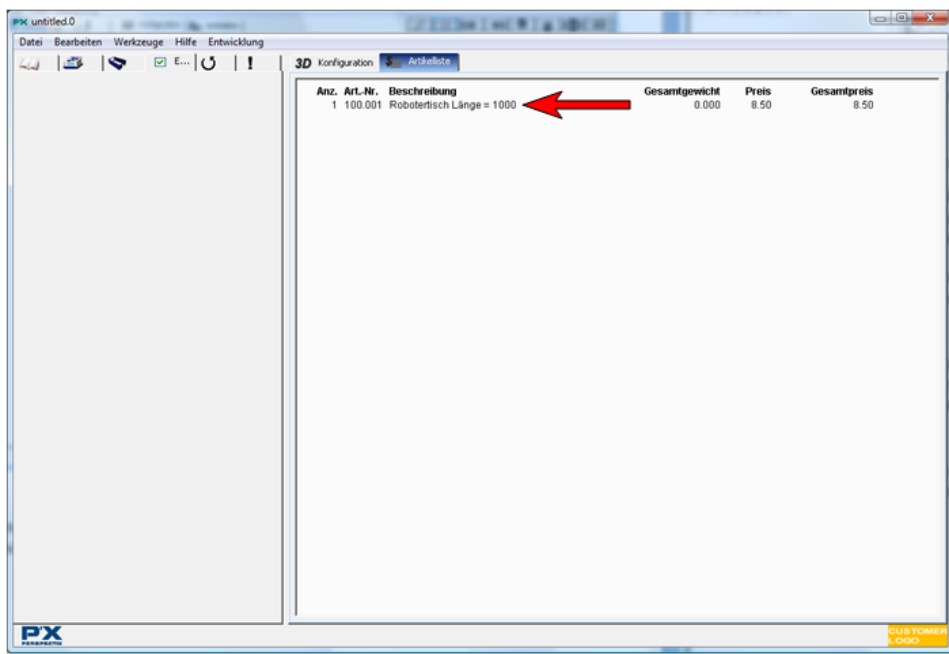
1. Start the **Configurator** and drag the robot table from the **Iconbar** into the **3D Scene**.



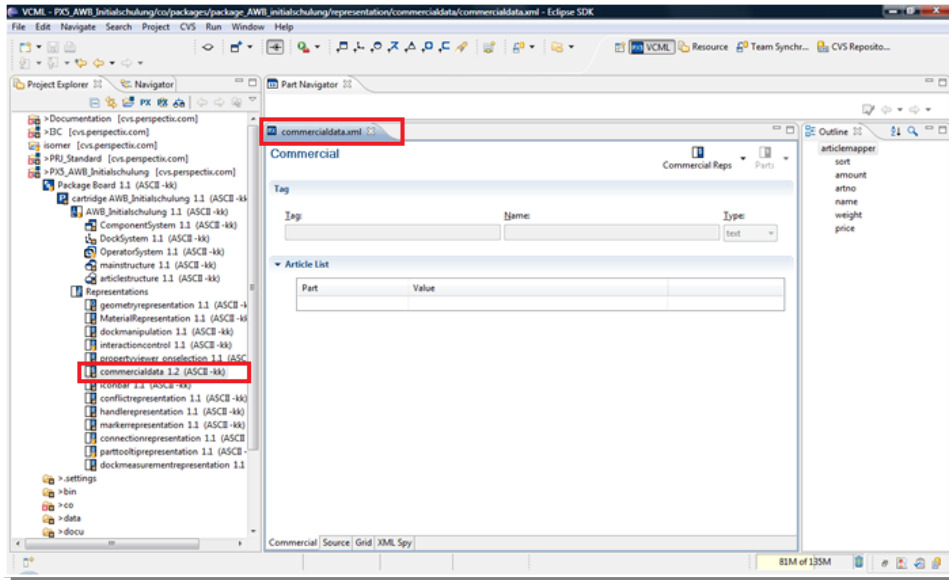
2. Click the **Properties** tab.
3. Select the robot table in the **3D Scene** and drag the slider to change the length of the table.



4. Open the **Article List**. The correct table length is not listed in the article list yet.

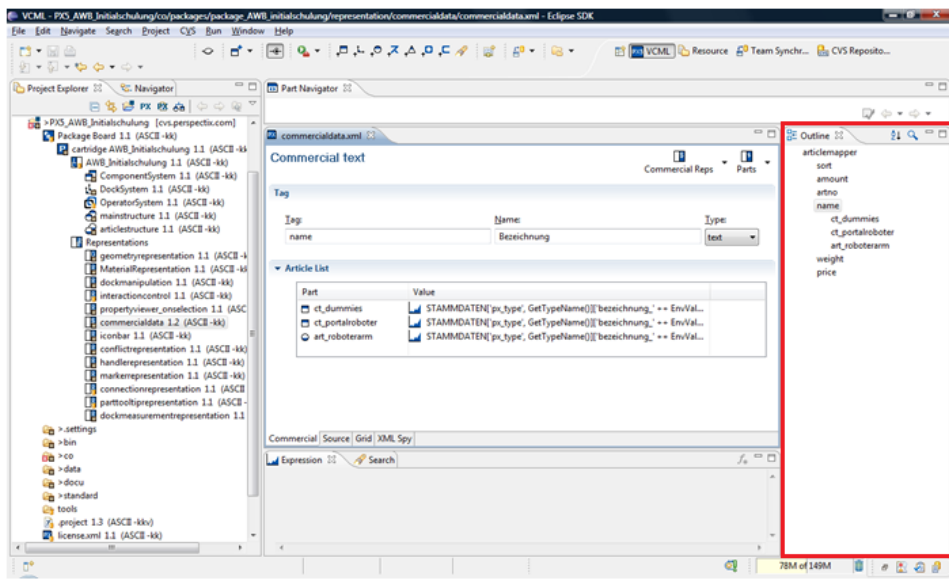


5. First, the correct table length should be listed in the description of the article. To do this, open the **Commercialdata Editor** (commercialdata.xml tab).

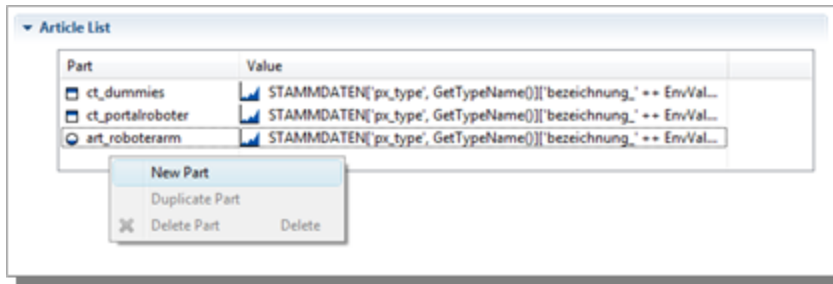


6. Take a look at the existing entries in the **Outline** of the **Commercialdata Editor**. Click the arrow to the left of the **articlemapper** entry to open the list. The arrow appears when you place the mouse cursor over the entry.
7. Open the **name** entry and take a look at the **Article List** table in the **Commercialdata Editor**.

This list defines that only those components, which are subordinate to the **ct_base** or **ct_dummies** component type, are listed, and that only article groups, which are subordinate to the **art_robot_arm** article type, are listed.



8. Right-click a field in the **Part** column in the **Article List** table and choose **New Part** from the context menu. A new row opens in the **Part** column.

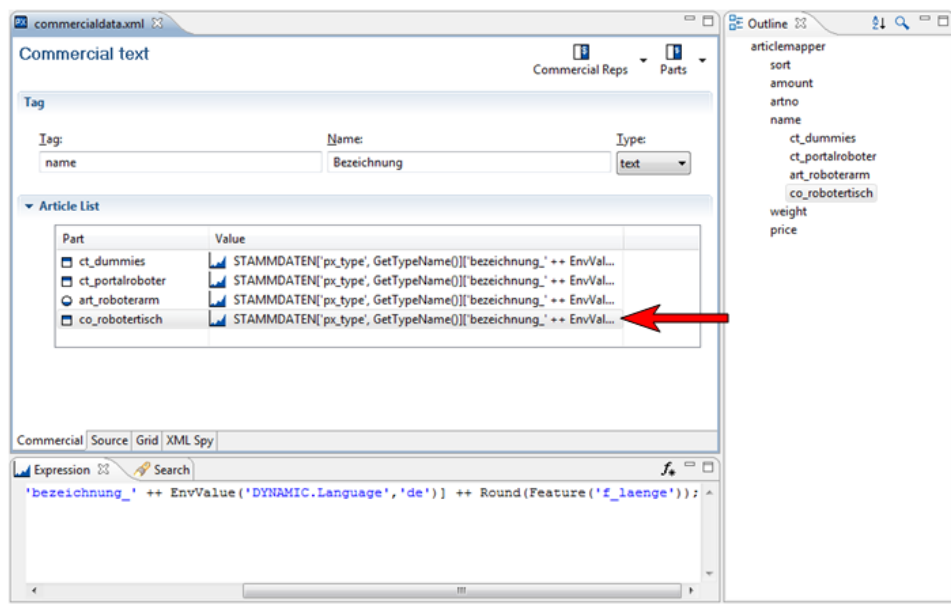


9. Type `co_table_var` in the new row or press CTRL+SPACEBAR and select an entry in the pop-up window.
10. Copy the `STAMMDATEN['px_type', GetTypeName()]['bezeichnung_' ++ EnvValue('DYNAMIC.Language', 'de')];` formula from one of the upper rows into the new row in the **Value** column.
11. Extend the copied formula as follows: `STAMMDATEN['px_type', GetTypeName()]['bezeichnung_' ++ EnvValue('DYNAMIC.Language', 'de')] ++ Round (Feature('f_laenge'));`

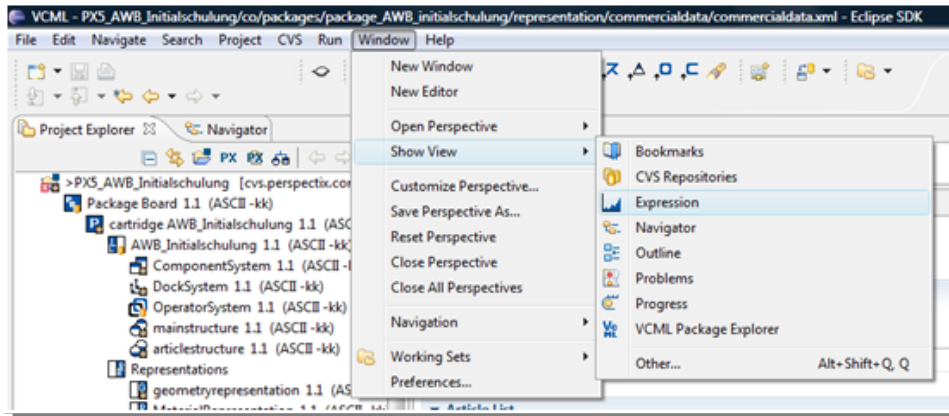
Note: Add the `++ Round (Feature('f_laenge'))` function at the end of the formula.

Caution: Add the function before the semicolon (;). The semicolon belongs at the end of the formula.

12. Save the **Commercialdata Editor**.



Note: To edit long formulas, choose **Window > Show View > Expression** from the main menu to open the **Expression View**.



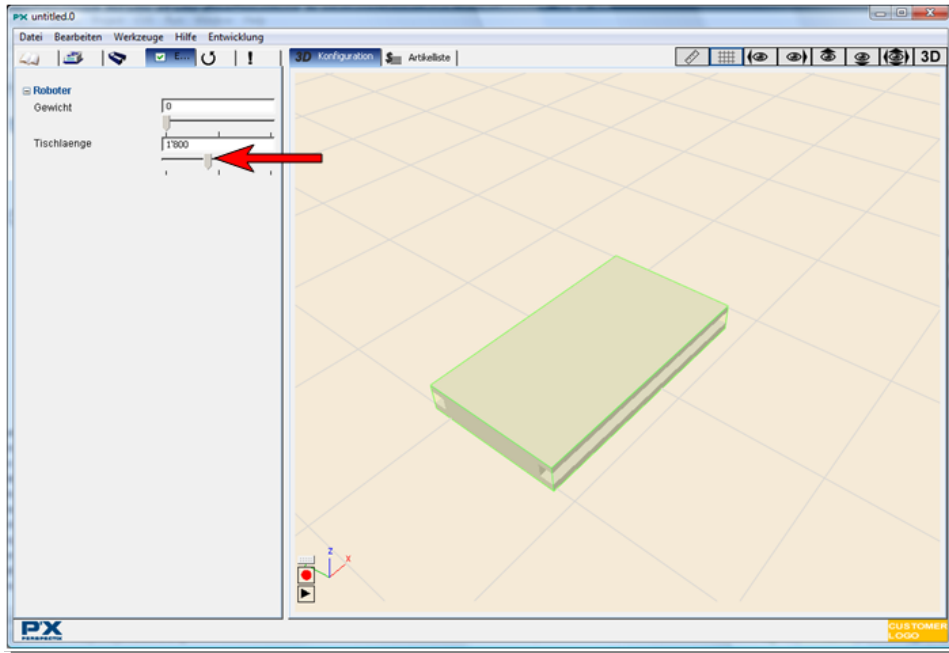
13. Right-click the `materialmaster.xml` file under `PX5_AWB_Tutorial/data/MasterData` and choose **Open with > Excel** from the context menu to open the file using Excel.
14. Change the `Robotertisch Länge = 1000` entry in the **bezeichnung_de** column to `Robotertisch Länge -`.
15. Change the English name next to it also.

Note: If you do not change the entry, the property of the correct length will simply be added to the number 1000 in the article list.

16. Save and close the file.

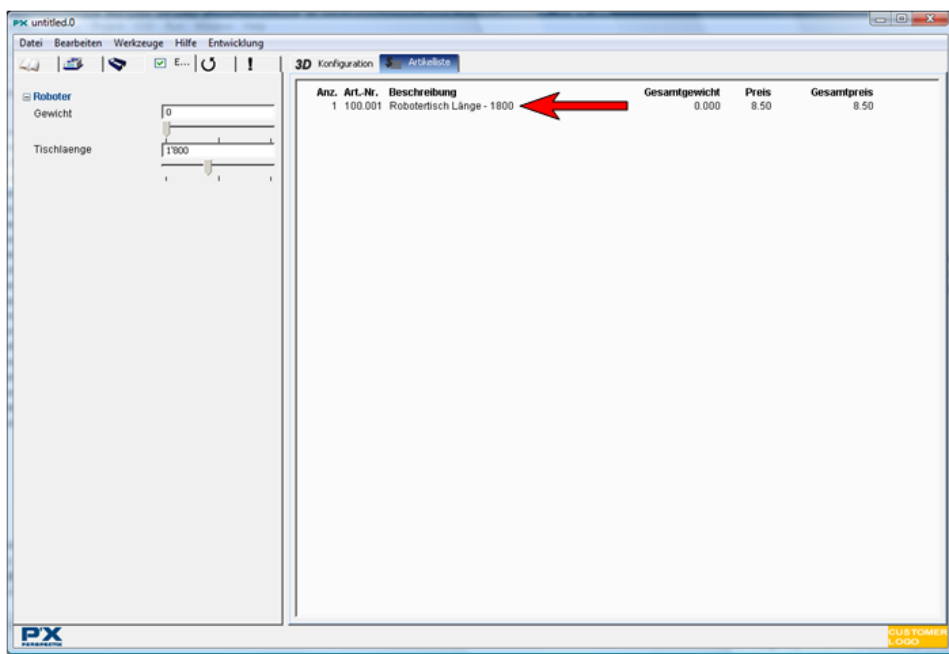
	A	B	C	D	E	F	G	H
	px_type	artikelnnummer	preis_ch	bezeichnung_de	bezeichnung_en	angebotstext_de	angebotstext_en	
3	co_robotertisch_var			Robotertisch Länge -	Robitable length -			
4	co_portal_za_1300	200.013	5555	Portal ZA-1300-zyl	Portal ZA-1300-zyl			
5		200.015		Portal ZA-1500-zyl	Portal ZA-1500-zyl			
6		250.013		Portal ZA-1300-quad	Portal ZA-1300-quad			
7		250.015		Portal ZA-1500-quad	Portal ZA-1500-quad			
8	co_roboteroberarm_zyl_200	300.013	466	Roboterarm KG-1300-zyl	Roboterarm KG-1300-zyl			
9	co_roboterunterarm_zyl_200	300.015	455	Roboterarm KG-1500-zyl	Roboterarm KG-1500-zyl			
10		350.013		Portal KG-1300-quad	Portal KG-1300-quad			
11		350.015		Portal KG-1500-quad	Portal KG-1500-quad			
12		400.001	4444	Roboterarm zylindrisch D200	Robiarm zyl. D200			
13	ar_robiarm_quad	500.001	3333	Roboterarm quadratisch	Robiarm cube			
14	co_bedienspanel_z99	999.099	2222	Bedienpanel 299	Userpanel 299			

17. Start the **Configurator** or if it still open, press F5 to update it. All necessary data will be reloaded.
18. Drag the robot table from the **Iconbar** into the **3D Scene**.
19. Click the **Properties** tab.
20. Select the robot table in the **3D Scene** and drag the slider to change the length of the table.



21. Open the **Article List**. The correct table length is now listed in the article list.

If the robot table is selected in the **3D Scene**, you can move the slider and the length of the robot table in the **Article List** will change.

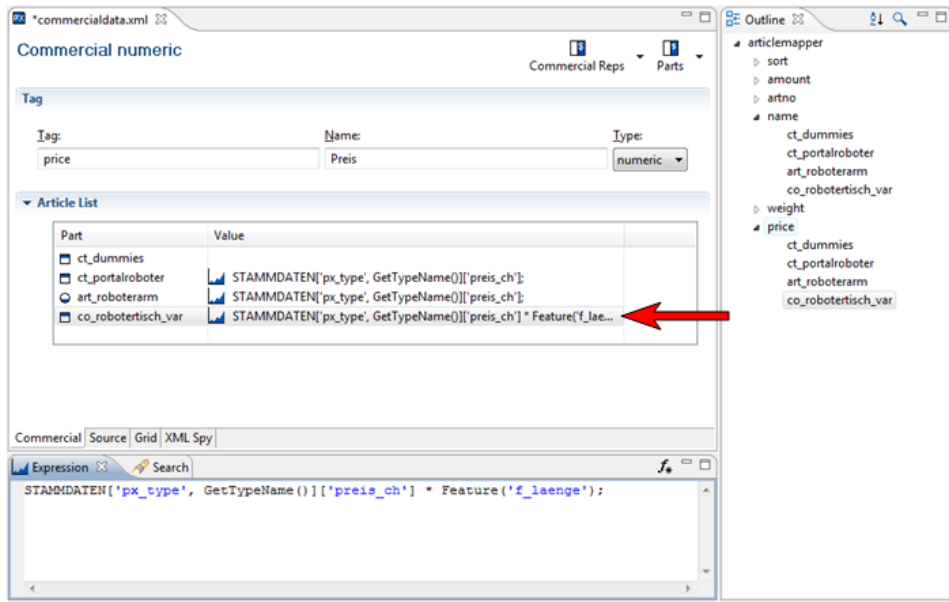


22. Now you must adjust the price of the robot table to the length in the **Article List**. To do this, go back to the **Commercialdata Editor** and select the `price` entry in the **Outline**.
23. Right-click a field in the **Part** column in the **Article List** table and choose **New Part** from the context menu. A new row opens in the **Part** column.
24. Type `co_table_var` in the new row.
25. Copy the `STAMMDATEN['px_type', GetTypeName()]['preis_ch'];` formula from one of the upper rows into the new row in the **Value** column.

26. Extend the copied formula as follows: `STAMMDATEN['px_type', GetTypeNames()]['preis_ch'] * Feature('f_laenge');`

Note: Add the `* Feature('f_laenge')` function at the end of the formula.

Caution: Add the function before the semicolon (;). The semicolon belongs at the end of the formula.

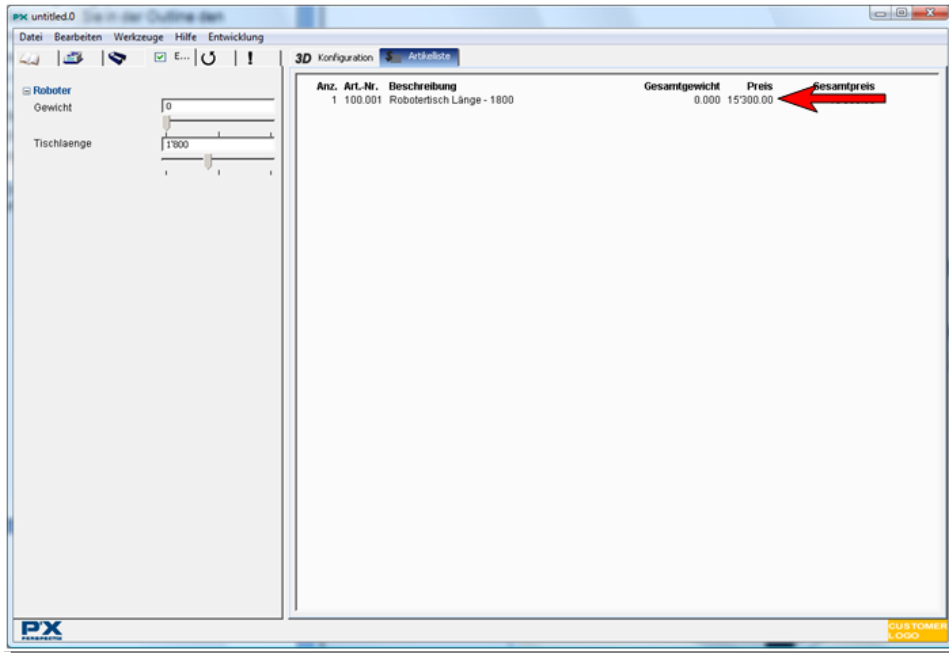


27. Save the **Commercialdata Editor**.

Result

1. Start the **Configurator** or if it is still open, press F5 to update it.

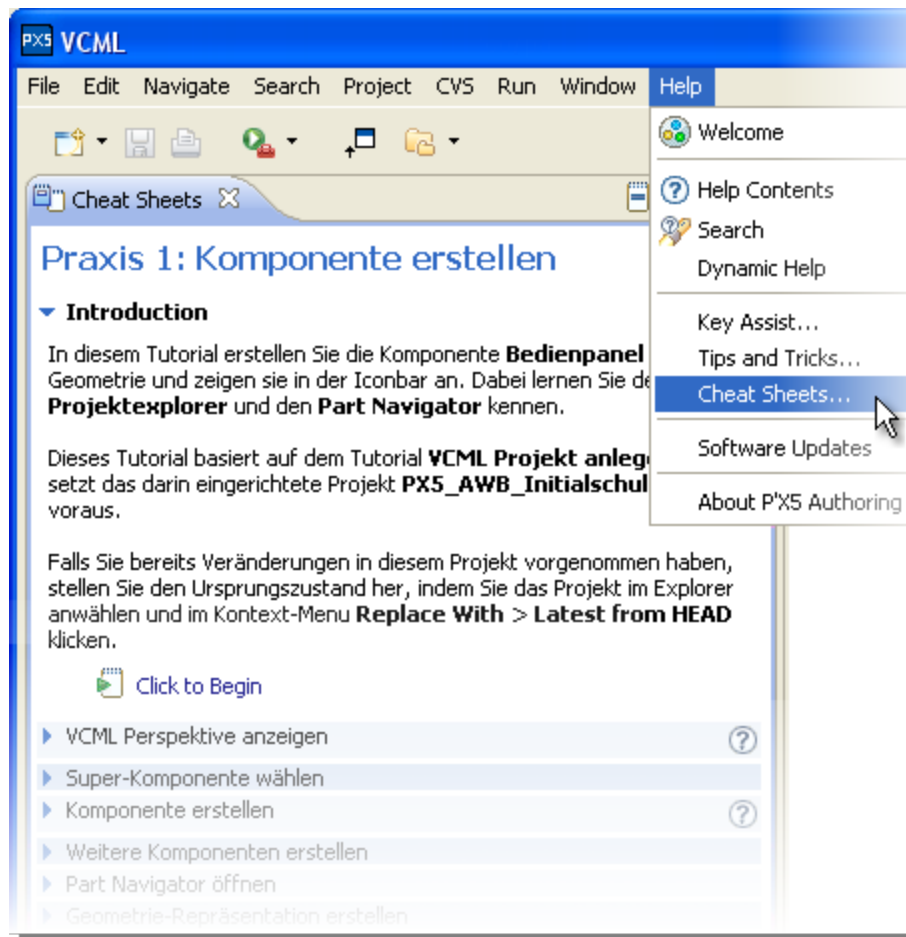
The price of the robot table has changed, i.e. the original price has multiplied with the length of the robot table.



2. If the price of the robot table seems to high, adjust the formula in the **Commercialdata Editor** as follows: `STAMMDATEN['px_type', GetTypeNames()]['preis_ch'] * Feature('f_laenge') / 100;` This divides the amount by 100.
3. Save the **Commercialdata Editor**.
4. Refresh the **Configurator** and compare the new price to the previous price.

Cheat Sheets

The Cheat Sheets are a set of tutorials closely modelled after the Basic Tutorial (Practice).



To open the Cheat Sheets

- Choose **Help > Cheat Sheets** from the main menu.

Geometry Requirements

Data Formats

The following data formats are acceptable:

- VRML (.wrl) version 1.0 (without the use of protos)
- JT
- FBX

Note: Make sure that the materialisation (it should be as authentic as possible, ideally without the use of textures and transparencies) is also correctly shown when exported. You can verify this using a VRML browser or JTOpen viewer.

Geometry Structure and Orientation

The geometry file should contain the entire part – provided it is not parametrizable anymore and the component parts of the part cannot be moved (e.g. doors and arms of machines).

The part should be positioned and rotated in the geometry file in such a way that the part stands "upright" on the "floor", i.e. the z-axis should point up and the part should stand with its real "base" on the x/y-level with z=0. Parts which are not positioned on the floor, should have their "natural" hanging/-placing point set on the z=0 level in the Properties tab.

Select the origin point in the geometry in such a way that it represents the natural attachment point to neighboring attachment parts at the same time (at least in one of the two coordinate axes).

Always create profiles so that a part of the profile is displayed with a 1 mm length and the origin point of a profile with a symmetrical cross section lies in the symmetry center on the "left" end of the profile. For profiles that are not symmetrical or are only symmetrical on one axis, you should also set the origin point on the "left" end of the profile, but on the "lower" symmetry axis or on a natural attachment point. For the orientation of the coordinate system, the positive z-axis should always point in the direction of the extrusion and the positive y-axis (for axis symmetrical profiles) should lie parallel to the symmetry axis.

Detailing and Visualisation Quality

The P'X5 Sales Solution is often used by sales people on laptops or computers outside the office. That is why it is necessary to "build" the geometries as light as possible, i.e. to reduce the assembly to only the most relevant information that a sales/service employee needs (the geometry should not contain any details that are not relevant for the configuration).

To ensure a high quality import of data into the P'X5 Sales Solution and to achieve good visualisation speed, the geometries that you import into the P'X5 Sales Solution must fulfill the following requirements.

Geometries that are not relevant are, e.g. spacers, screws, casing embossments, small bores, etc. However, you must consider how the object looks when you reduce the geometry. The goal is to make the object look as real as possible, while at the same time keeping it as reduced as possible.

Depending to the usage scenario, you should reduce the number of polygons to a minimum:

- Larger parts (e.g. machines, robots, etc.), of which there are not many in a configuration, should have 2000 to 5000 polygons.

- Smaller parts (e.g. bases, terminals, motors, other attachment elements), which occur often, but whose geometric details are usually not very relevant for the configuration, should have 200 to 500 polygons.
- Complex parts, which are important configuration elements (especially if the geometric details make information about the selected option visible) should have between 300 and maximum 1000 polygons.

Docks

Docks (snapping points) are used to connect and build parts. Define the position of the docks (in x/y/z-coordinates relative to the origin point of the geometry file), as well as the rotations around the three axes.

The best way to do this is to create a UserCoordinateSystem (UCS) in the CAD system or to set a targeted point. Its position and rotation in relation to the geometry origin point must be read and transferred together with the geometries.

In most cases, at least one of the dock points will coincide with the origin point or will be out of alignment in only one of the two axes.

A transport element can be used as an example:

- As described above, the origin point of the geometry should lie on the entry side in the center of the conveyor line at the contact point with the upstream transport element, but on the ground level.
- This way the dock point for the entry side of the transport element in the x- and y-direction can be identical with the origin point and is only displaced upwards in the z-direction.
- The corresponding dock point for the exit side would also be identical in the y-direction with the origin point, and would be displaced in the x-direction by the length of the transport element, in addition to the displacement in the z-direction (for a horizontal transport element).
- A rotation of the dock point on the exit side would only be necessary for a curve element. Here the y-value of the dock point is of course different and the dock point should be rotated around the z-axis, so that an identical transport element can be correctly attached with its entry side to the exit side, so that the dock points (exit dock of the first transport element and entry dock of the second transport element) lie on top of each other in position and rotation.

Note: Generally, you should carry out a short specific requirements and concept definition together with the authoring team of the P'X5 application. If it is clear what kind of parts will be used and how they will be connected to the parts already implemented in P'X5, the authoring team of the P'X5 application can create a requirements and concept definition beforehand, without consulting the providers of the geometries.

Collision Geometries

If a collision check is to be carried out in the Configurator, you should provide even simpler bounding box geometries (collision area of the part).

For geometrically simple parts, which will not be connected to other parts, P'X5 can semi-automatically create a collision box (based on the bounding box). You must decide and later verify on an individual basis if the collision box is sufficient.

Common Preferences

When installing an Eclipse-based tool, all preferences are set to default. In a team environment, or when upgrading to a new version, you might want to load existing preferences. Also, you might want to have default values for your team/organization, but still be able to customize for your own instance of Eclipse.

Eclipse provides a functionality for importing and exporting preferences. However, it only allows for full export of all preferences. The basic functionality of the Common Preferences plugin is to have someone create a preferences file using the preferences export, place it in a specified place, and then configure users' Eclipse instances to read this file at startup. The main usage is likely in a team environment, when a tool administrator will manage the preference file, but it can also be used in a single user environment.

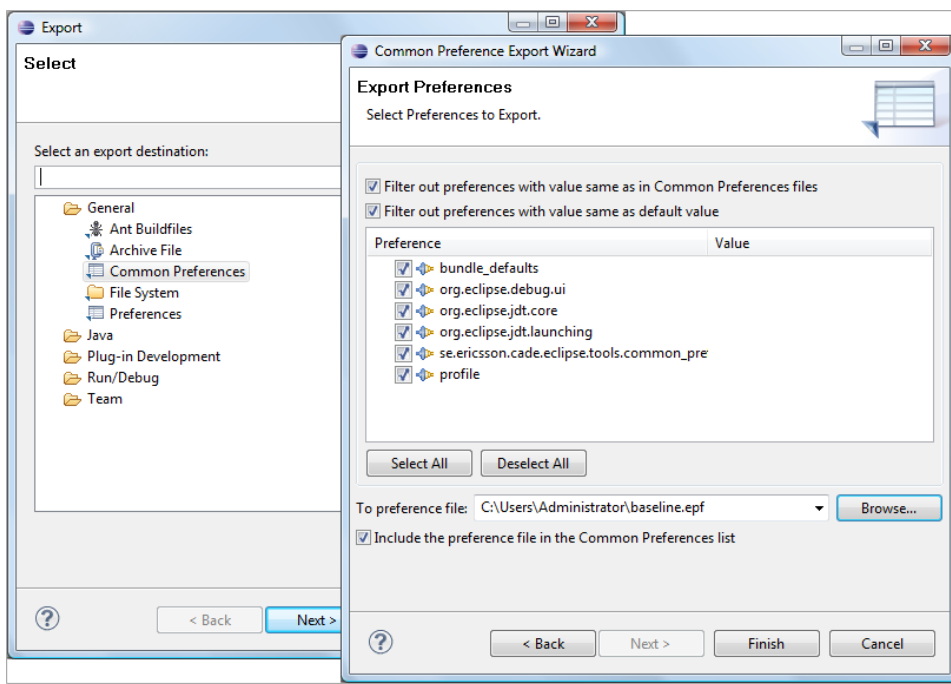
First, you need to export the preferences. Then, you can configure the preferences.

Export Preferences

The preferences to export are listed in the Export Preferences window of the Common Preference Export Wizard.

To open the Common Preference Export Wizard

1. Choose **File > Export** from the main menu.
2. In the **Export** window, choose **General > Common Preferences** and click **Next**.



The preferences are grouped based on the respective plugin they belong to. This makes it a little difficult to understand exactly what preference setting you want to export, especially if you have done many changes to your preferences. Hence, it is good practice to either start Eclipse with a new workspace or create a "preference baseline" before changing any of the default preferences you want to export.

To export preferences

1. Create a preference baseline using the **Export Preferences** window. Select all preferences and click **Finish** to save them to a file (e.g. `baseline.epf`). Creating a preference baseline is optional, but recommended for the reason mentioned earlier.

Make sure that the **Include the preference file in the Common Preferences list** checkbox is selected. This will enable you to filter out these settings in the next step.

2. Set the preference values you want to export using the **Preferences** window (choose **Window > Preferences** from the main menu). After you are done making changes to the preference values, click **OK**.

3. Export the changes you made to a preference file using the **Export Preferences** window.

Each individual value is not selectable, because a plugin might generally need to keep its saved values consistent. Select one or multiple values, and click **Finish** to save them to a file (e.g. `changed_prefs.epf`).

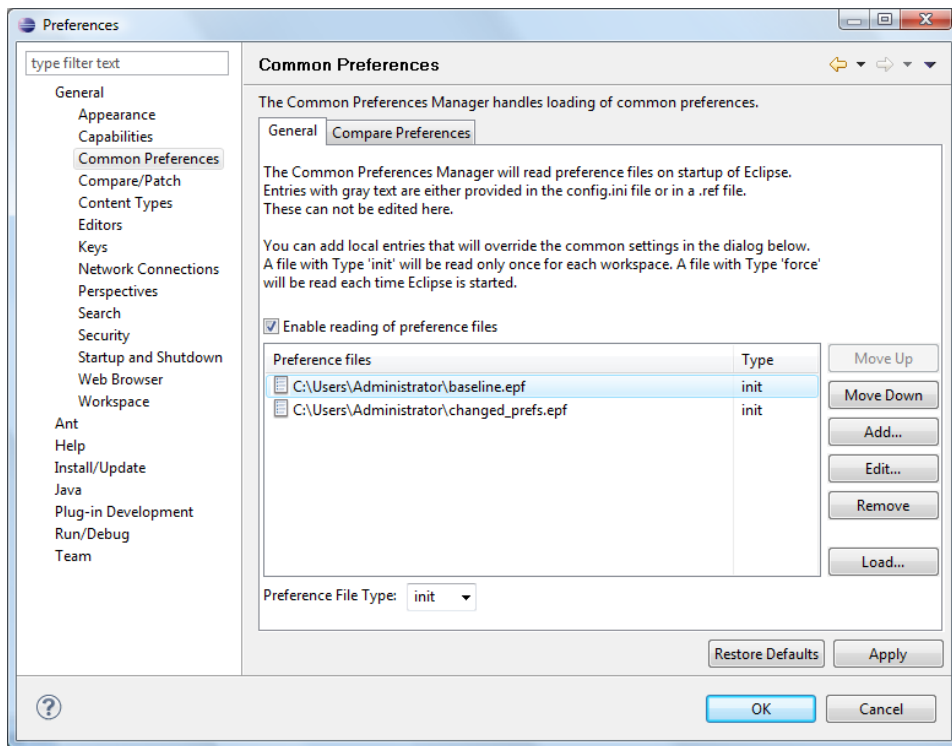
4. If you want to use the file on your machine, save the file anywhere on the disk. If you want to share the file, save the file to a location accessible to all clients. Both URLs (i.e. `http://changed_prefs.epf`) and UNC paths (i.e. `\\ComputerName\SharedFolder\changed_prefs.epf`) are supported as locations.

Configure Preferences

Once the desired preferences have been exported and a preference file is available, either at an URL location or on a file system, users' Eclipse instances can be configured.

To configure preferences

1. Choose **Window > Preferences** from the main menu and then **General > Common Preferences** in the **Preferences** window to open the **Common Preferences Manager** window.



2. From the file list, select the preference file you want to load the preferences from.

To add a file to the list, click **Add**, type the location of the preference file, and click **OK**.

3. For the selected file, select "init" (recommended) or "force" in the **Preference File Type** list box.
 - A file with the type "init" will be read only once for each workspace.
 - A file with the type "force" will be read each time the Authoring Workbench is started.
4. Click **Load** to load the preferences.
5. Click **Yes** to proceed and then **OK** to finish.

Export VCML Documentation

You can create a stand-alone and read-only documentation output of a VCML cartridge.

This documentation serves as reference information for other authors, and allows third parties to review the structure of the cartridge, i.e. it serves the same purpose that javadoc does for Java.

First, you need to set up the Infocenter. Then, you can export the documentation.

Set Up Infocenter

Infocenter is the application serving the HTML help to clients (typically browsers). It is a headless Eclipse application based on Eclipse 3.5 (Galileo), plus a custom plugin that contains the help content produced by the VCML documentation exporter.

To set up Infocenter

1. Download the Infocenter package for Windows from <http://www.perspectix.com/px5editor/download/infocenter.win32.win32.x86.zip>, or for Linux from <http://www.perspectix.com/px5editor/download/infocenter.linux.gtk.x86.zip>.
2. Extract the zip file to a location of your choice, called *<folder>* below.

Note: If unzipping fails because the path names are too long, try extracting to a short path, e.g. `C:\infocenter`.

3. Run the *<folder>/infocenter/infocenter.exe* file.
4. Type `http://localhost:3040/help/index.jsp` into your browser to display the Infocenter content.

Note: Windows (or another firewall) may ask if it should allow a server running on port 3040. To change the port, edit: *<folder>/infocenter/infocenter.ini*

Note: The opening console may display the message "MESSAGE Warning: no update sites configured!". Don't worry about that - the self-update feature allows a deployed Infocenter to update the documentation contents from an update site. To hide the console, remove `-consolelog` from *infocenter.ini* and end the application using the Task Manager.

Application Content

The application contains a host of folders and files - don't worry about that. The documentation content under *<folder>\infocenter\plugins\com.example.vcml_1.0.0* is the only thing you should care about.

The plugin contains the following static folders and files:

- `css/` The CSS files defining the style of the HTML documentation.
- `icons/` Icons needed for the tree structure.
- `js/` The javascript file for toggling inherited information display.
- `META-INF/` Plugin metadata.
- `plugin.xml` Plugin metadata.
- `vcml_1.0.0.xsl` A copy of the built-in XML transformation stylesheet.

In addition, the VCML documentation export creates the following elements:

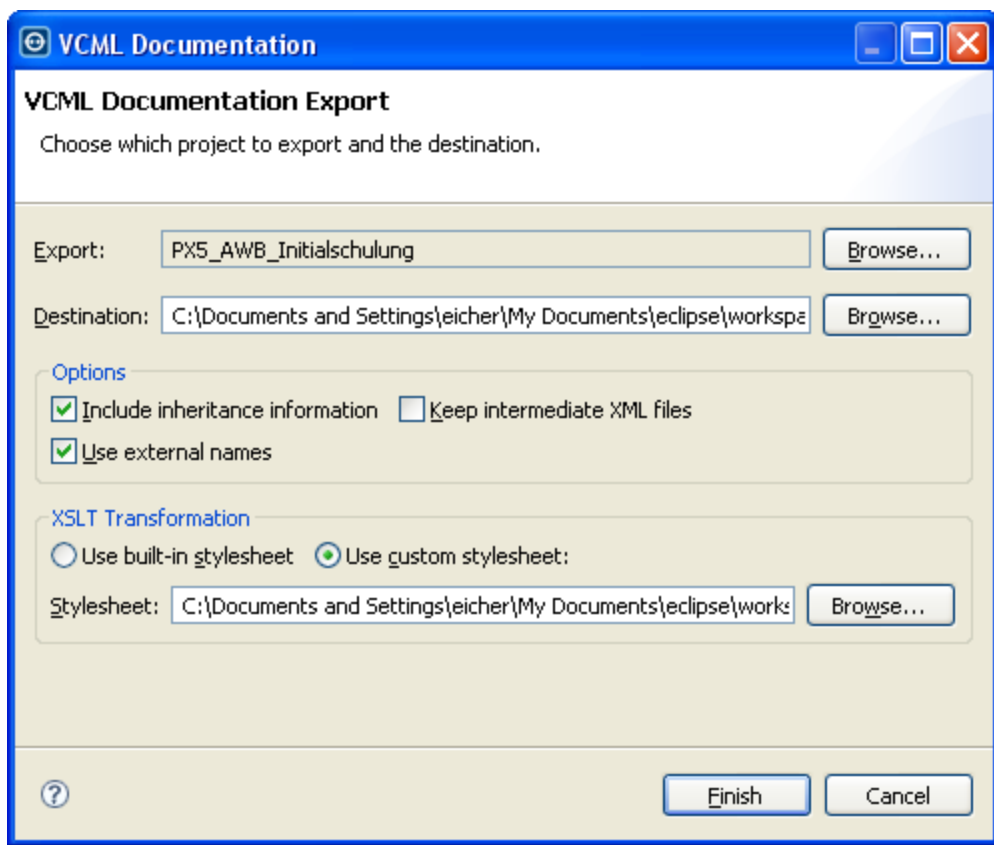
- PROJECT_NAME/ The actual documentation content as HTML files.
- index.toc The table of contents file, which the Infocenter uses to create the tree structure.

Export the Documentation

Note: To export the documentation, you need a license. Contact your contact person at Perspectix to obtain the correct license.

To export the documentation

1. Right-click the project in the **Project Explorer** and choose **Export VCML Documentation** from the context menu.
2. In the **Destination** field, select the `<folder>\infocenter\plugins\com.example.vcml.doc_1.0.0` folder. This is where the content is stored on the disk.
3. Do one of the following:
 - To use the built-in stylesheet, which is found in the `vcml.doc.xsl` file in the folder mentioned above, select the **Use built-in stylesheet** checkbox and select the corresponding location in the **Stylesheet** field.
 - To use a custom stylesheet, select the **Use custom stylesheet** checkbox and select the modified version in the **Stylesheet** field.



4. To ensure that Infocenter picks up the changed contents after exporting the documentation, close the Helpcenter (press CTRL+C in the console window), restart it, and refresh the browser to see the changes.

Stylesheet and CSS Hacking

The CSS should be self-explanatory. The `vcml.doc.xsl` transformation stylesheet is used to transform the intermediate XML representation of the exported project into HTML.

Tip: When debugging, select the Keep intermediate XML files checkbox in the VCML Documentation window.

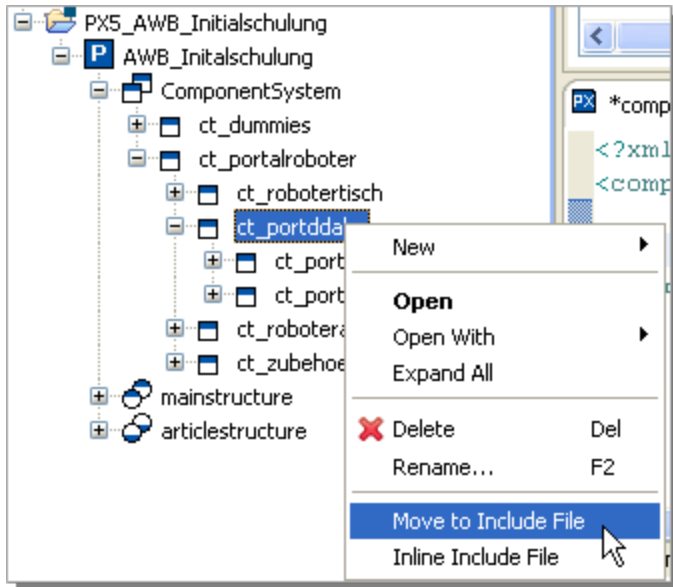
The stylesheet is structured after the different kinds of exported elements, such as properties, components, assemblies. Many common parts, such as assigned properties for components and assemblies are called as named templates.

To map internal to external names, an `xsl:key` element is used, which references the `PROJECT_NAME/px_externalnames.xml` file created by the export. The URL of that file is passed as a parameter to the transformation by the VCML Documentation window.

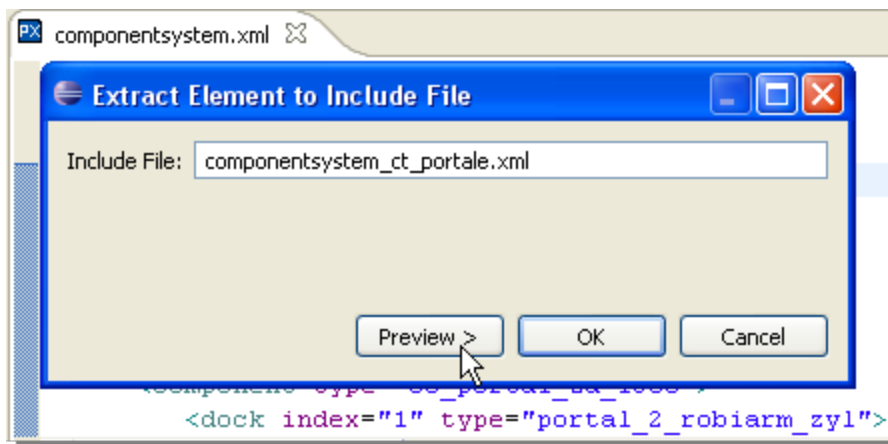
Manage Include Files for Parts

To manage include files for parts

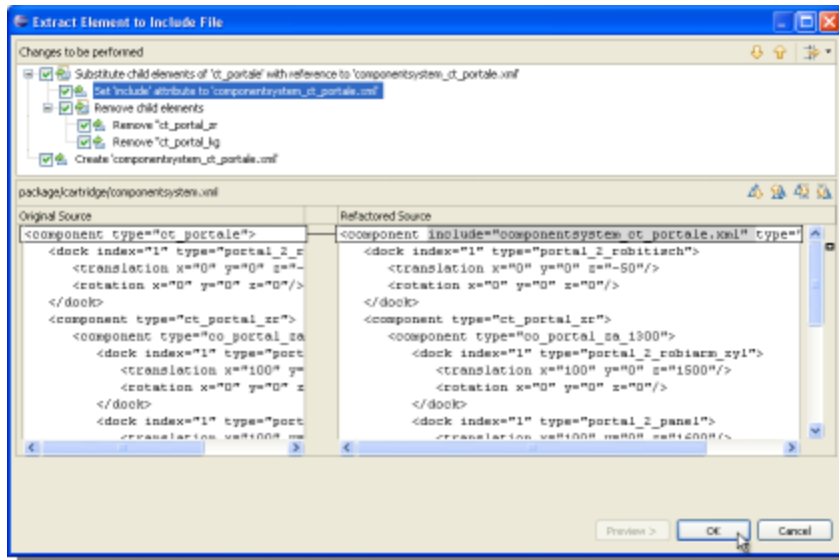
1. Right-click an entry in the **Project Explorer** and select **Move to Include File** or **Inline Include File** from the context menu.



2. Type the name of the include file or accept the proposed name.



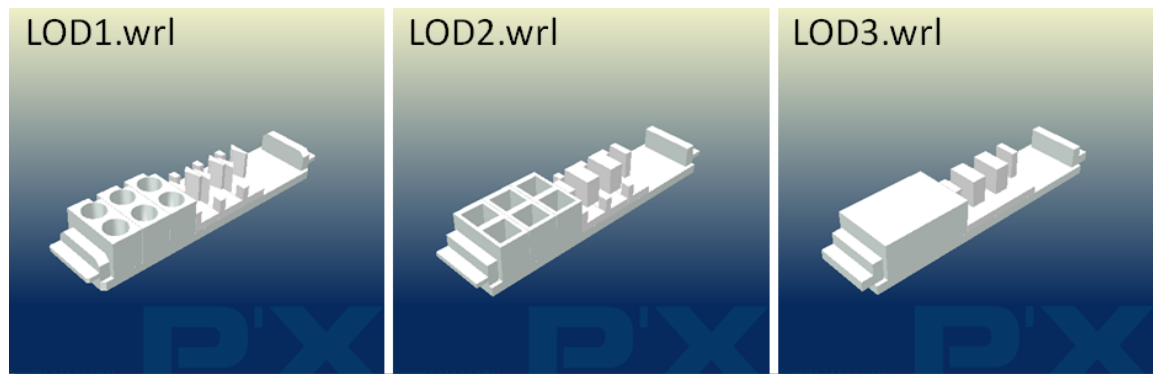
3. Click **Preview** to display a preview of the proposed changes.



Merge LODs with Polyedit

The following example shows how a 3D file is created with Levels of Detail (LODs). Levels of Detail are different gradations of detail that are used when displaying virtual worlds.

The following three WRL files have different detailing levels. These files will be placed on top of each other in Polyedit, the file with the lowest detailing first and the file with the highest detailing last, and saved as one 3D file. Depending on the zoom level that you set, the corresponding file will be displayed.

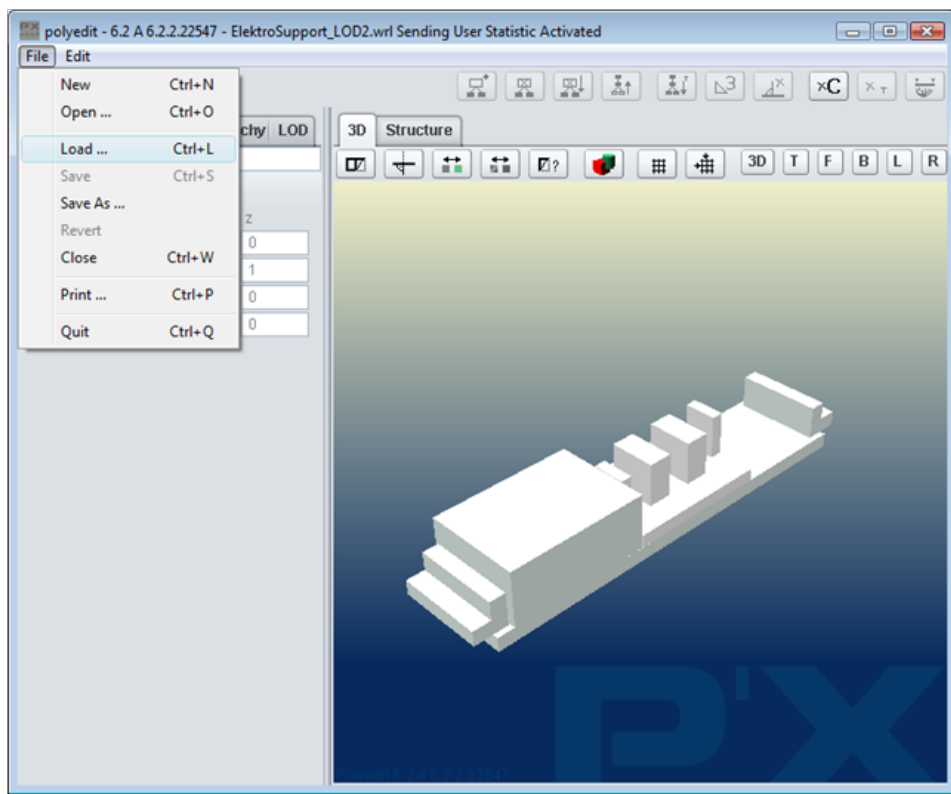


Prerequisites

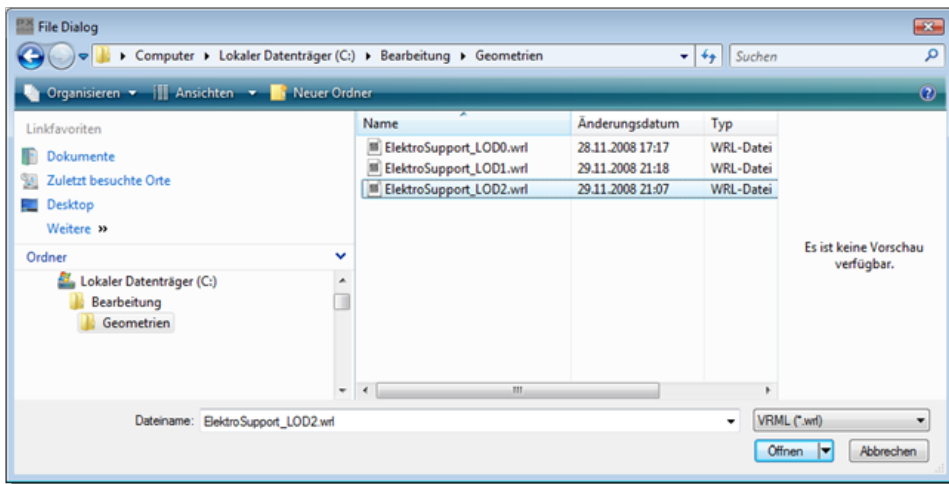
- The origin point of the geometry files must lie on the same place.

To create a 3D file with Levels of Detail (LODs)

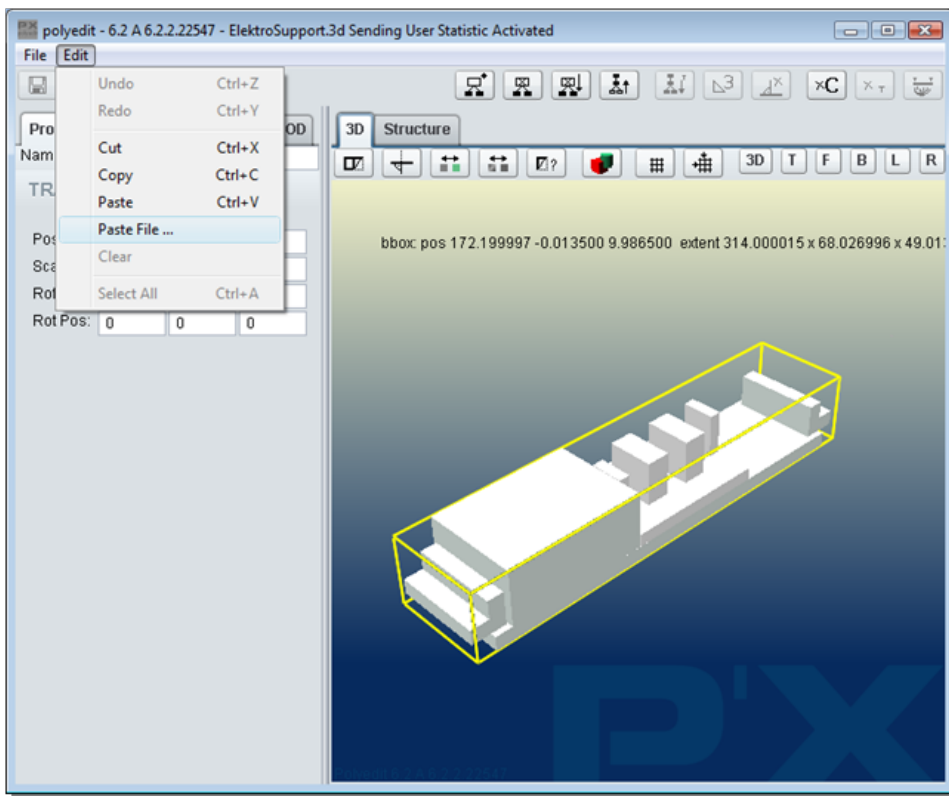
1. Start Polyedit.
2. Choose **File > Load** from the main menu and load the first geometry file (i.e. the file with the lowest detailing).



3. Choose **File > Save As** from the main menu and save this file as *.3d.



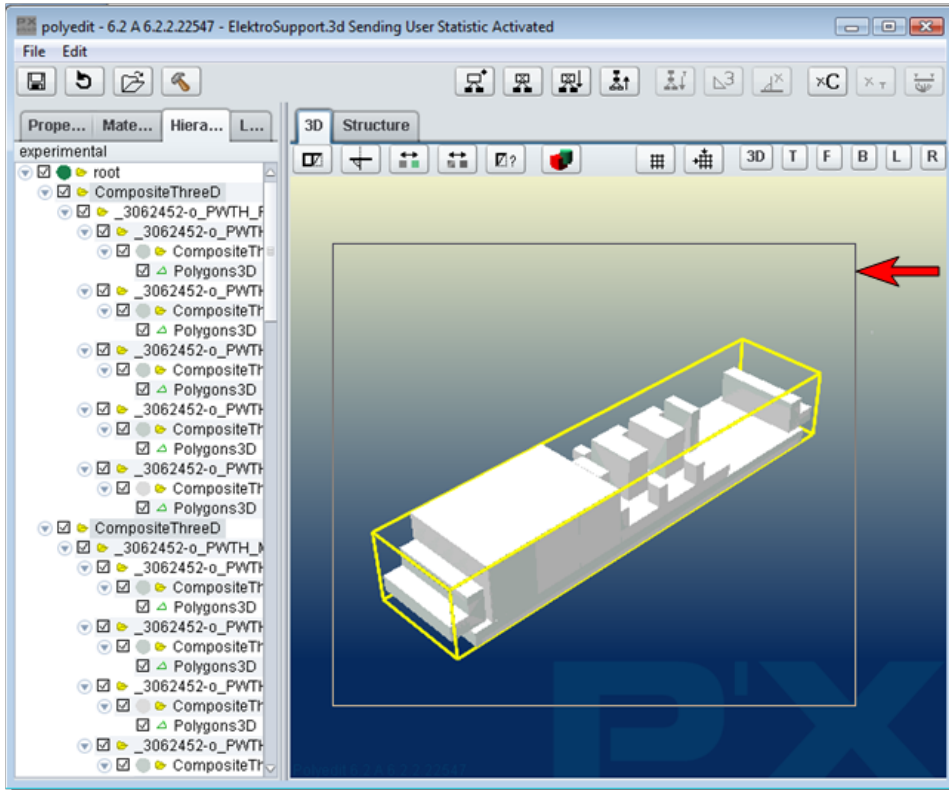
4. Click the loaded geometry and choose **Edit > Paste File** from the main menu to load the geometry with the next highest detailing. The new geometry file is placed above the already loaded geometry. The two geometry objects will lie on top of each other, as long as the coordinates have not been changed.



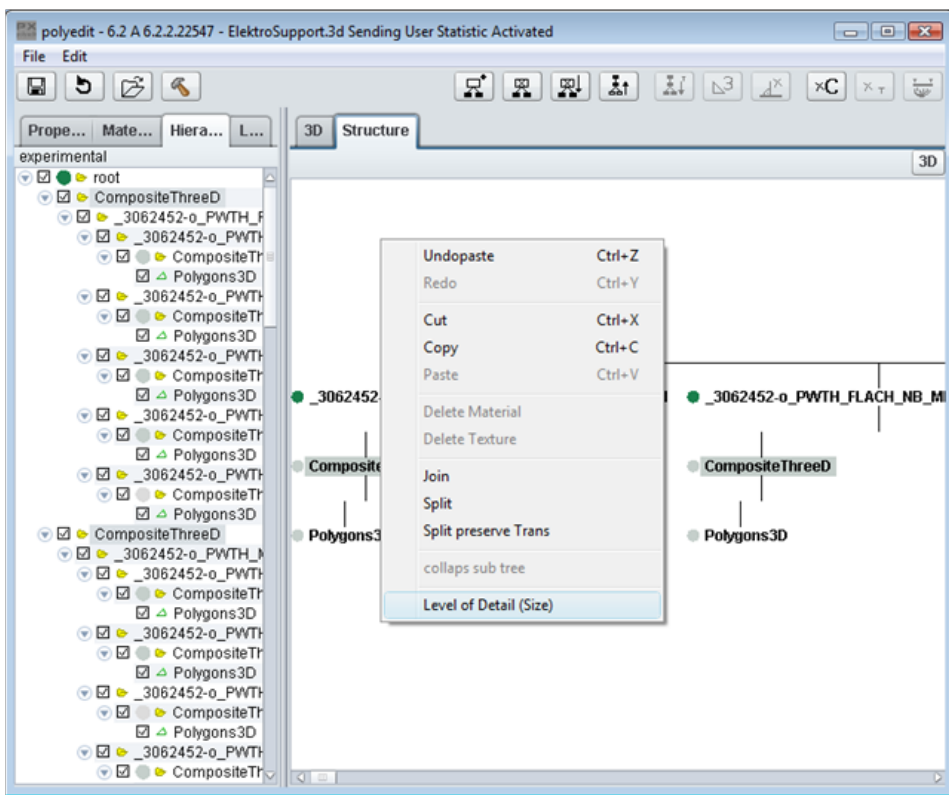
5. Repeat step 4 for every other geometry, which represents an LOD, one after another from the lowest detailing to the highest detailing.

Caution: Make sure you keep the order of the detailing (from the lowest to the highest).

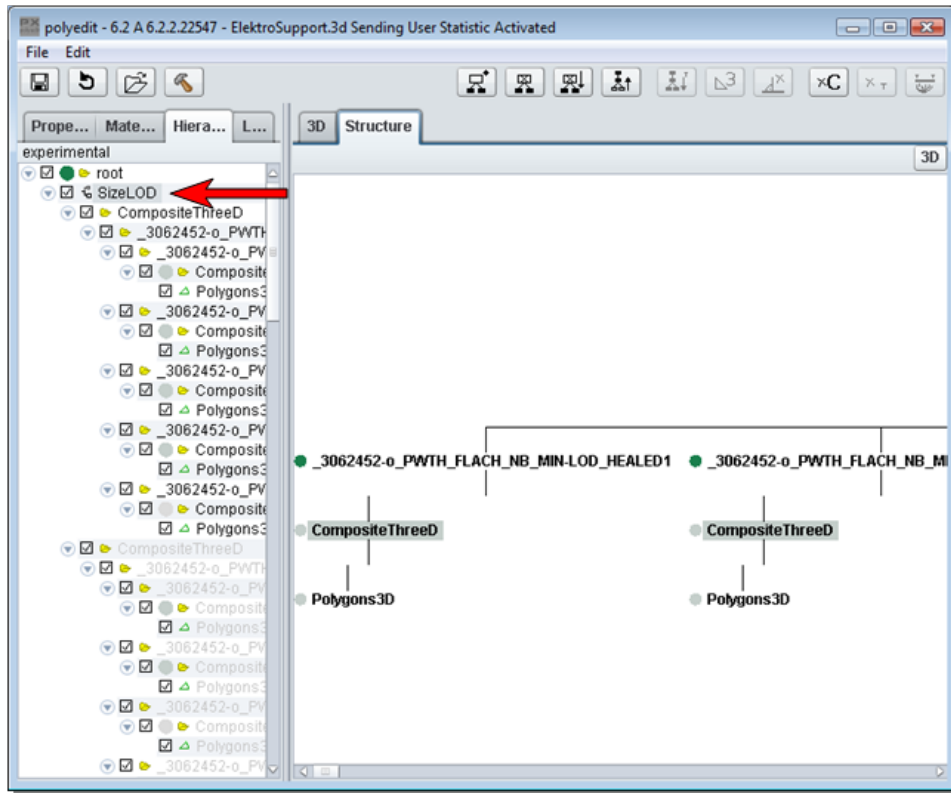
6. Open the **Hierarchy** tab and select (by drawing a selection rectangle around all objects) all loaded geometry objects in the **3D View**.



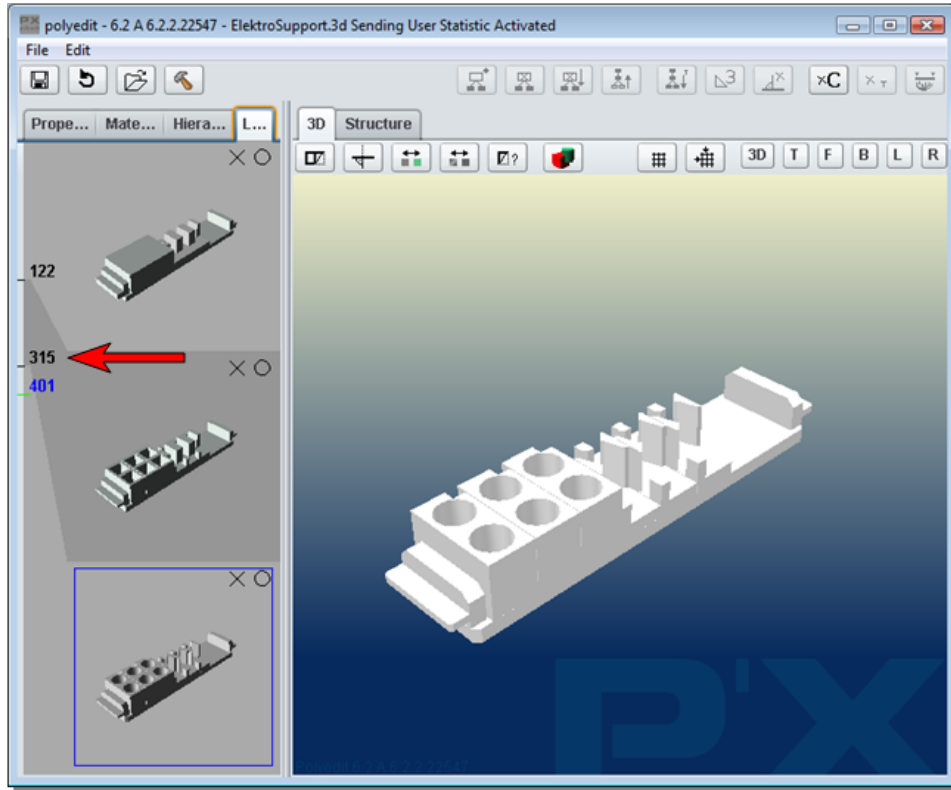
7. Open the **Structure** tab. Right-click the white background and choose **Level of Detail (Size)** from the context menu. The nodes in the **Hierarchy** tab should still be selected. A LOD file with the *.3d ending is saved.



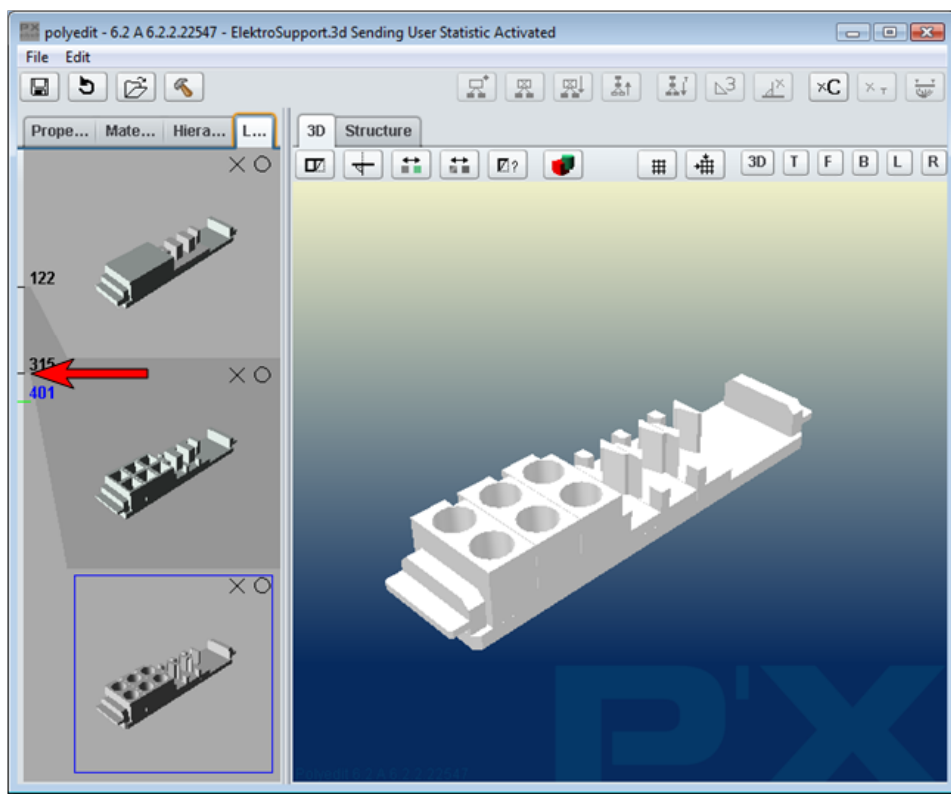
8. A new node called "SizeLOD" is created under "root" and above the selected node in the **Hierarchy** tab.



9. Open the **3D** tab (right) and the **LOD** tab (left). The **LOD** tab shows the different geometries. The geometries are separated by a separator and a black number. The black number is the depth value, at which one image switches to another image when zooming (2D Bounding Box diagonal of the screen display in pixels).

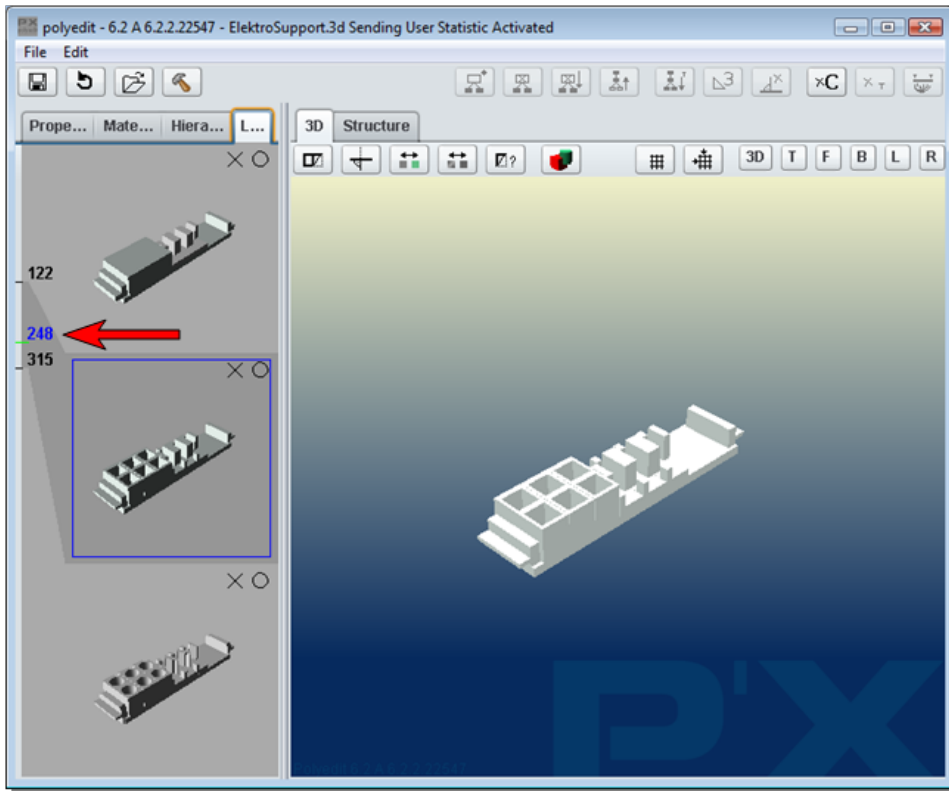


10. To change the depth value, move the black separator.

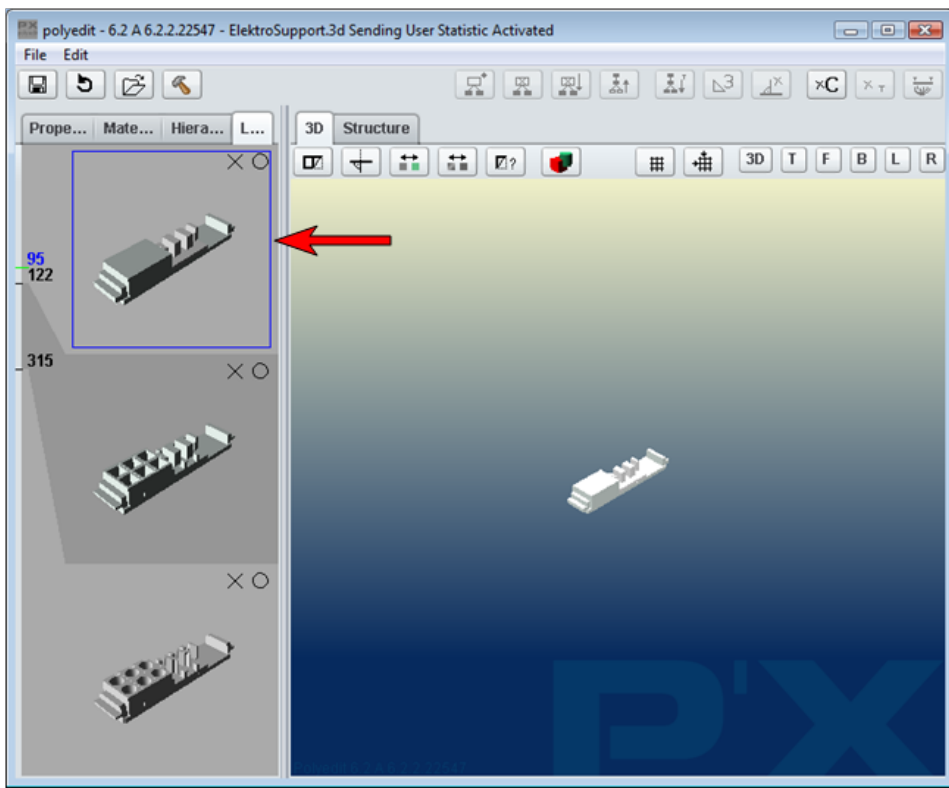


11. Switch back to the **3D View** and scroll the mouse wheel. When you change the zoom, the blue number changes. The blue number is the zoom value in the 3D display. Now, zoom the 3D

object to the place (display size) at which one display level switches to the next display level. Drag the black separator to this place in the **LOD** tab.



12. The blue frame shows which detail level is shown in the **3D View**. The graphic display in the **3D View** changes, when the appropriate separator is moved beyond the blue number.



Polyedit License Handling

The Polyedit application needs a valid license for operation. It therefore looks for a license file containing the following entry:

```
<feature
  application="polyedit"
  feature="polyedit"
  activated="true"
  param=""
  date="a valid date"
  hash="a valid hash"
  ...
/>
```

Note: Refer to your contact at Perspectix for a valid license file.

To indicate the location of the license file to Polyedit

Do one of the following:

- On the command line, start Polyedit with the parameter: `-LIC: polyedit.exe -LIC c:\Perspectix\license.xml`
- Define the `PXLicenseFile` environment variable, which contains the absolute path to the license file: `set PXLicenseFile= c:\Perspectix\license.xml`

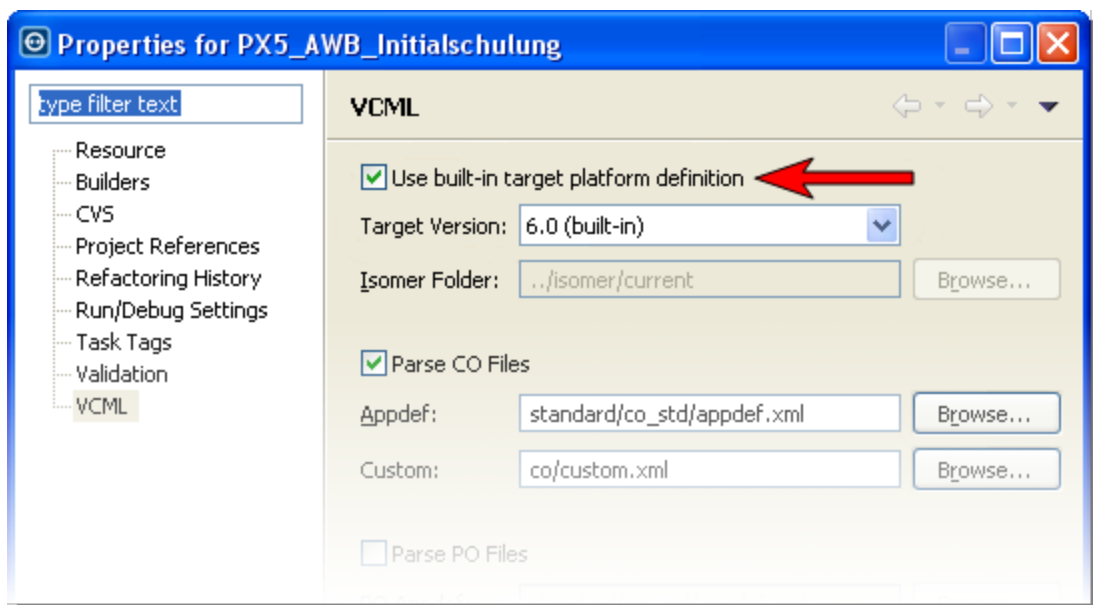
Select Built-In Target Platform

P'X5 projects can be built against a built-in target definition, instead of referencing a project in the workspace. This simplifies the management of multiple projects referencing different versions, and reduces one source of errors when setting up a new project.

A target platform defines the expression functions, named operations, operators, and environment variables available in a specific version of the P'X5 application and accompanying files.

To enable a target platform

1. Right-click a P'X5 project in the **Project Explorer** and choose **Properties** from the context menu.
2. Click **VCML** in the **Properties** window.
3. Select the **Use built-in target platform definition** checkbox and select a target version in the **Target version** field.



Note: You can also install old and unsupported target platform versions.

Refactor Expressions

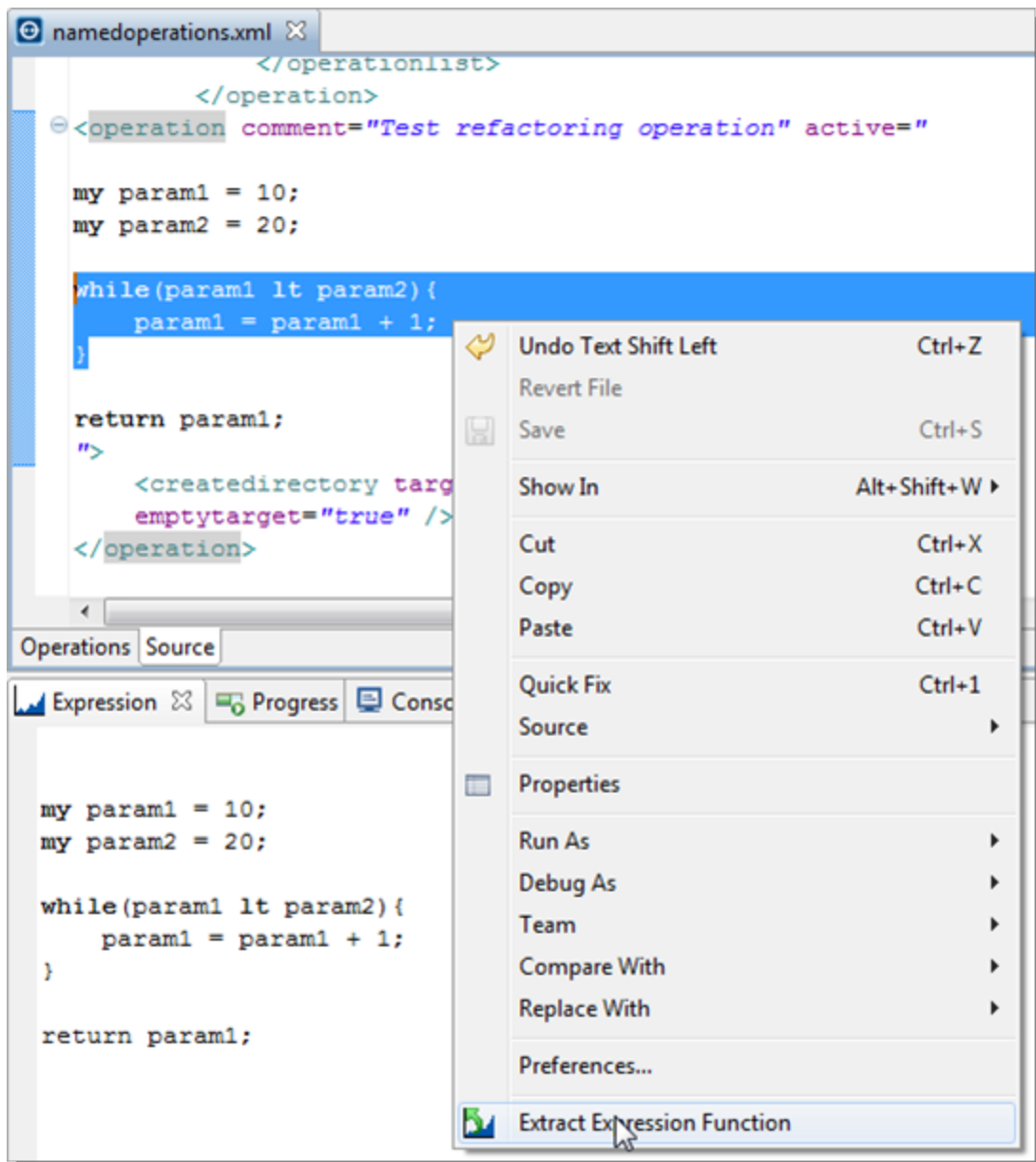
Version: Requires P'X5 version 7.0 or higher.

Extract Expression Function

You can extract parts of an expression and create a function from it.

To extract a part of an expression

1. Select or mark the expression code (on the **Source Page** or in the **Expression View**) that you want to extract.
2. Right-click the code and choose **Extract Expression Function** from the context menu.



3. Enter the data in the **Extract Expression Function** window and click **OK**.

Note: You only have to enter a name, all other fields are optional.

The dialog box is titled "Extract Expression Function". It contains the following fields and sections:

- Name:** A text field containing "myNewFunction". A callout points to the "Final" checkbox, stating: "Declares the extracted function as final."
- Final:** A checked checkbox.
- Appcode File:** A text field containing "co/packages/tutorial/cartridge/appcode.xml" with a "Browse..." button. A callout points to this field, stating: "Extracts the function directly into the defined application definition file (appdef.xml). Otherwise the function is created inline."
- Comment:** A text area containing "This is the comment of the refactored function." A callout points to this area, stating: "Value that is returned by the function."
- Returnvalue:** A section containing:
 - Type:** A dropdown menu set to "Double".
 - Description:** A text area containing "This is the description of the return value". A callout points to this area, stating: "You can rename, move, delete and add parameters."
- Parameters:** A table with columns "Name", "Description", and "Type".

Name	Description	Type
param1	This is parameter 1	double
param2	Description of second param	double
newParameter	We can even add a new one!	Object

To the right of the table are buttons: "Add", "Remove", "up", and "down".
- Preview:** A text area showing the generated function signature: `def final myNewFunction (param1,param2,newParameter)`. A callout points to this area, stating: "Shows a preview of the new function signature."
- Buttons:** "Preview >", "OK", and "Cancel".

4. After the refactoring is done, the function is extracted and the previously selected text is replaced with the call of the new function.

```

</operationlist>
</operation>
<operation comment="Test refactoring operation" active="
my param1 = 10;
my param2 = 20;

call myNewFunction(param1,param2,newParameter);

return param1;
">
  <createdirectory target="${dir}/"
    emptytarget="true" />
</operation>

```

5. If you entered an application definition file (appdef.xml) in the **Extract Expression Function** window, the function is created there with the entered metadata.

```

<function description="This is the comment of the refactored function.">
  <param description="This is parameter 1" name="param1" type="double" />
  <param description="Description of second param" name="param2" type="double" />
  <param description="We can even add a new one!" name="newParameter" type="Object" />
  <returnvalue description="This is the description of the return value of this function."
    type="Double" />
  <definition expression="
def final myNewFunction(param1,param2,newParameter){
  while(param1 lt param2){
    param1 = param1 + 1;
  }
};" />
</function>

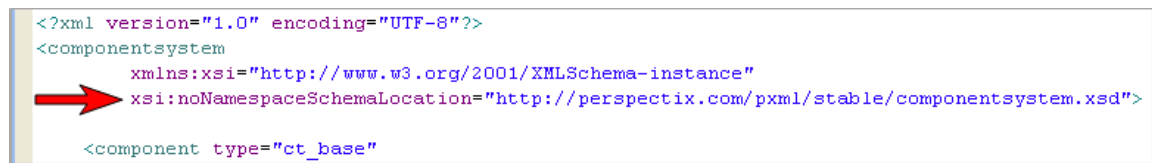
```

Replace PXML Schemas

The target platform definitions provide information about a specific P'X5 version and contribute an XML schema catalog to validate the PXML files.

Currently, PXML projects typically validate against the schemas in a P'X5 project in the workspace, by specifying a relative `noNamespaceSchemaLocation` attribute. To take advantage of the built-in schemas, these locations must be changed to reference an absolute location. The PXML schemas reside below the `http://perspectix.com/pxml/` location.

Perspectix recommends that you validate against the version of the PXML schemas corresponding to the project's target platform. To do so, the schema location URL should look like in the following example:



```
<?xml version="1.0" encoding="UTF-8"?>
<componentsystem
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://perspectix.com/pxml/stable/componentsystem.xsd">
  <component type="ct_base"
```

To replace relative schema locations common in a PXML project

- Replace this:

```
:noNamespaceSchemaLocation\s*=\s*"(\.\.[/\\])+isomer[/\\]current[/\\]schema[/\\]
```

With this:

```
:noNamespaceSchemaLocation="http://perspectix.com/pxml/stable/
```

Translate User Interface Language

The user interface (menu entries, button names, etc) of the P'X5 Sales Solution is available in German, English, Italian, French, Spanish and Russian. When you customize the application you can translate the standard languages to other languages.

Translation Process

To translate the user interface language, you must:

1. Define the languages.
2. Add a new language.
3. Add all terms, which have been added by the customization.
4. Translate the terms.
5. Switch the language in the Project Organizer.

Caution: All files in the `standard` folder may not be externalized. The terms in the `standard` folder are translated by Perspectix. If other languages are added to the standard languages, which means that the terms in the `standard` folder must be translated, the following steps must be performed:

1. Copy all entries in the standard language (e.g. German) from the `//standard/shared_std/gui/guilanguages/gui_languages_de.xml` file into the file of the customized area `//data/ApplicationData/gui/guilanguages/gui_languages_de.xml`.
2. Only translate the languages that were added to the standard languages in the Translation Editor.

To define the languages

1. Add the `guilanguages=" ../data/ApplicationData/gui/guilanguages.xml"` attribute to the `<arcoapplicationdefinition>` element in the `//co/custom.xml` and `//po/custom.xml` customization files. In the `guilanguages.xml` file you define which languages are used by the customization.

```
<arcoapplicationdefinition
...
  guilanguages=" ../data/ApplicationData/gui/guilanguages.xml"
>
```

To add a new language

1. Add another row with the language definition and file assignment to the `//data/ApplicationData/gui/guilanguages.xml` file, e.g.:

```
<?xml version="1.0" encoding="UTF-8"?>
<languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="../../../../isomer/current/schema/languages.xsd">
  <language langabrev="de" langname="deutsch" file="guilanguages/gui_language_de.xml"/>
  <language langabrev="en" langname="english" file="guilanguages/gui_language_en.xml"/>
  <language langabrev="es" langname="spanisch" file="guilanguages/gui_language_es.xml"/>
  <language langabrev="fr" langname="français" file="guilanguages/gui_language_fr.xml"/>
  <language langabrev="it" langname="italiano" file="guilanguages/gui_language_it.xml"/>
  <language langabrev="ru" langname="russian" file="guilanguages/gui_language_ru.xml"/>
  <language langabrev="sr" langname="serbian" file="guilanguages/gui_language_sr.xml"/>
</languages>
```

2. Copy the `gui_language_sr.xml` file referenced above (in the same format as the other language files) to the `//data/ApplicationData/gui/guilanguages` folder.

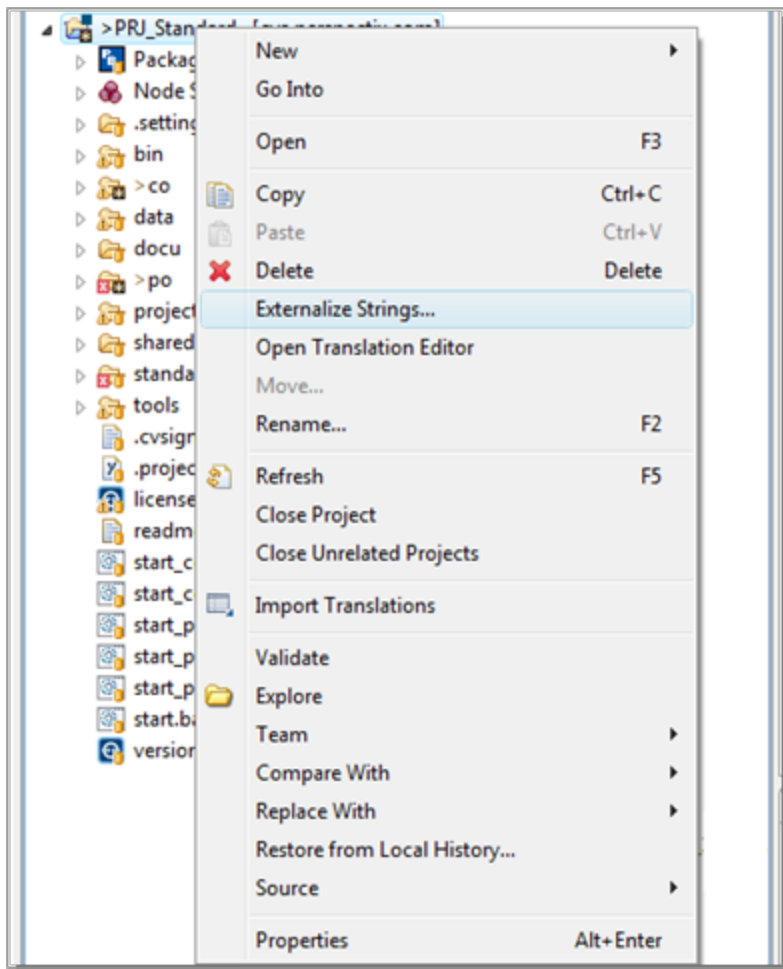
Note: Watch the filename endings (e.g. `_str`). Each language has its own file name ending, which must be the international language identification.

To add all terms that have been added by the customization

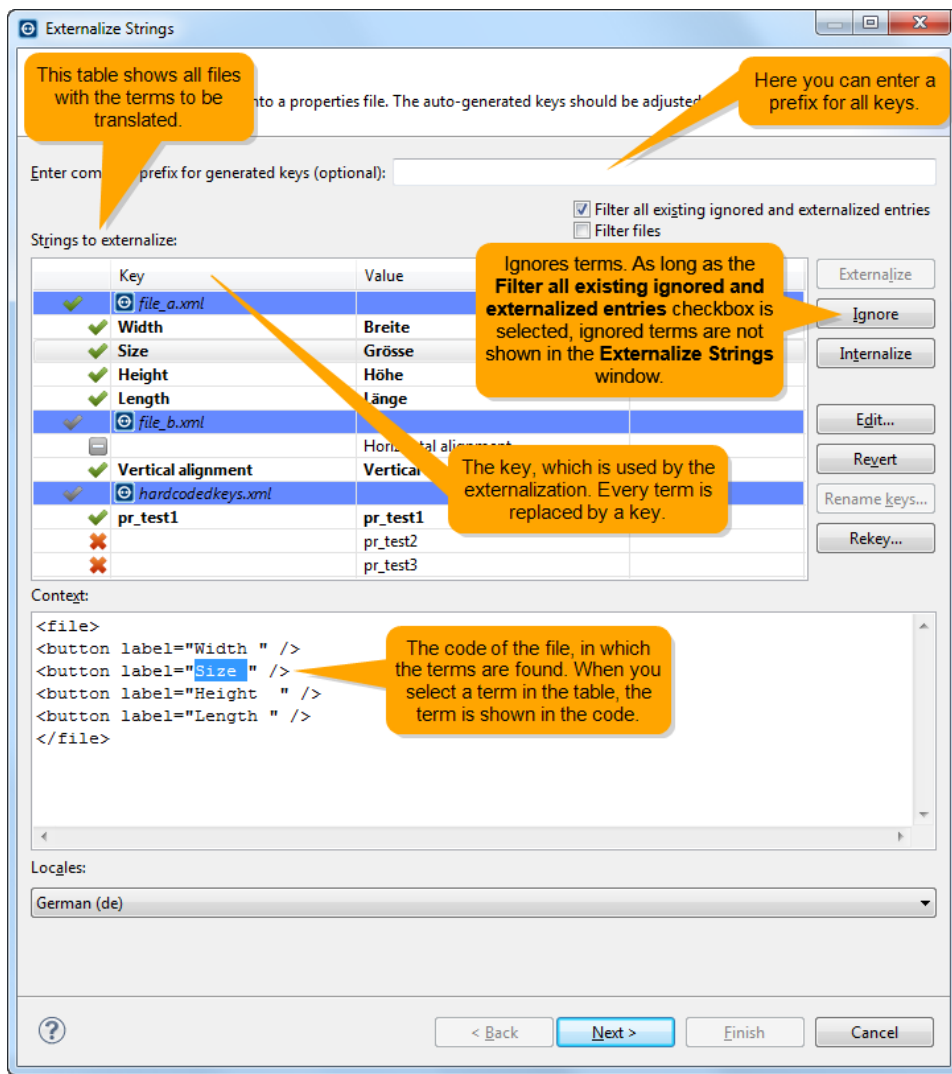
This includes all terms for properties, buttons in the iconbar, markers in the 3D Scene, etc. The materials are excluded and are defined in the material master.

1. Right-click the project in the **Project Explorer** in the **Authoring Workbench** and choose **Externalize Strings** from the context menu.

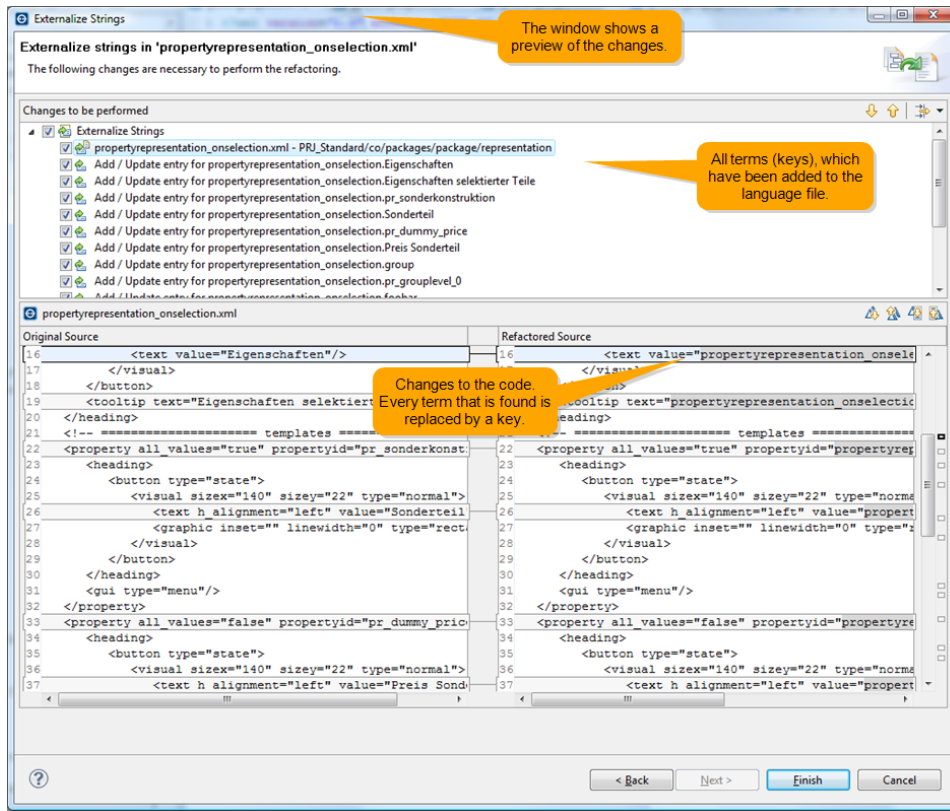
Note: If only one file contains terms, which must be translated, click that file and not the project.



2. In the **Externalize Strings** window, select the files you want to externalize and click **Externalize**. When you have externalized all the files you want, click **Next**.

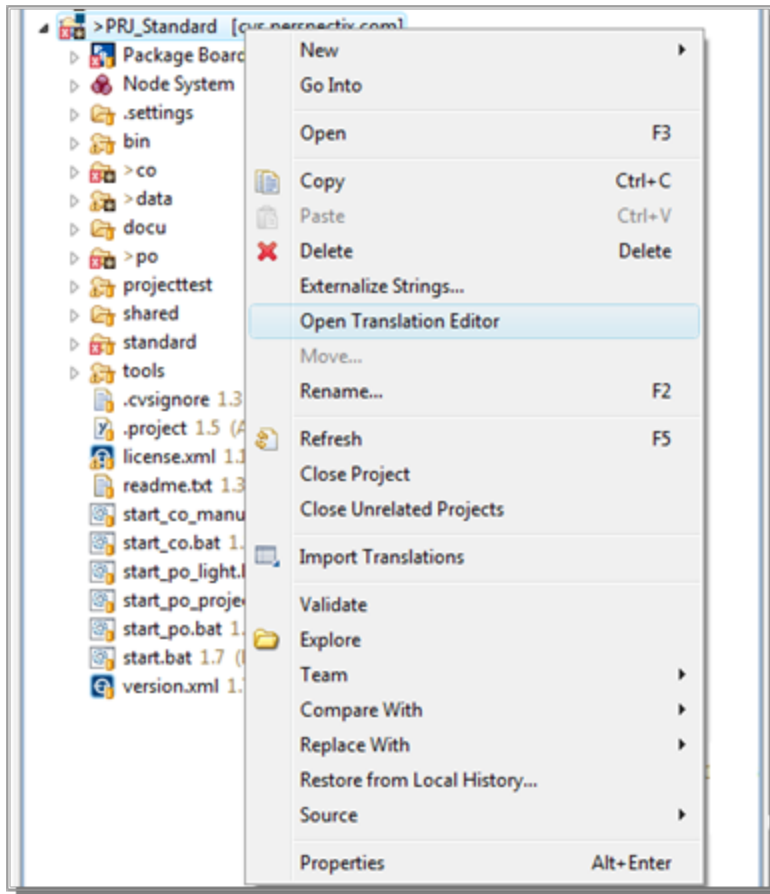


3. Click **Finish** to add the terms to the language file.

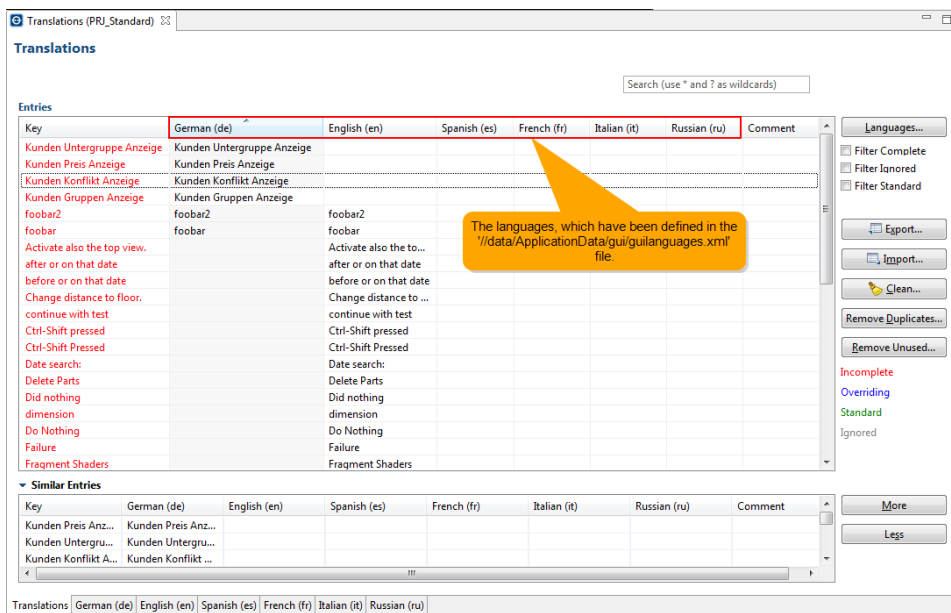


To translate the terms

1. Right-click the project in the **Project Explorer** in the **Authoring Workbench** and choose **Open Translation Editor** from the context menu.



2. Either translate the terms directly in the **Translation Editor** or click **Export** to save the terms in a text file.



The text file is tab-separated and can be opened with Excel.

Note: Not all applications save data in the same way in the CSV format (e.g. Excel uses two types of data separation characters).

Tip: Save tables in the Unicode text format (*.txt), which uses tabs as data field separators and saves data in the Unicode format (UTF-16).

Note: Excel does not support line breaks in data fields. Versions older than Office 2007 might support even less functions.

3. If you exported the terms to a text file and translated them, click **Import** in the **Translation Editor** to import the text file with the translated terms.
4. Save the translation. The translated terms are written in every defined language file.

To switch the language in the Project Organizer

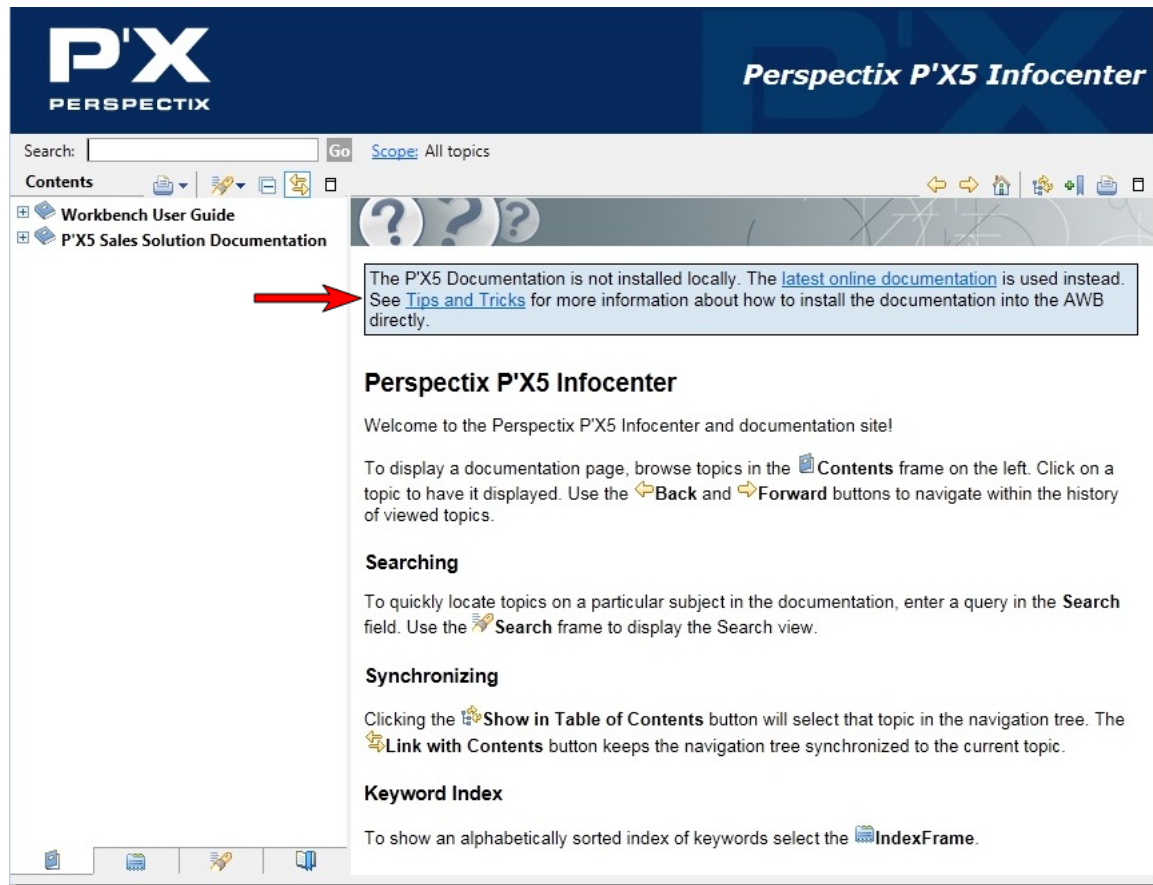
1. Choose **Administration > Language** from the main menu.
2. Select a language in the **User Language** list box in the **Users** tab in the **Administration** window.

Tips and Tricks

Click any link below to see tips and tricks.

Install P'X5 Documentation

The P'X5 Documentation is NOT included in the Authoring Workbench by default. A message appears at the top of the welcome page when you open the Help Contents.

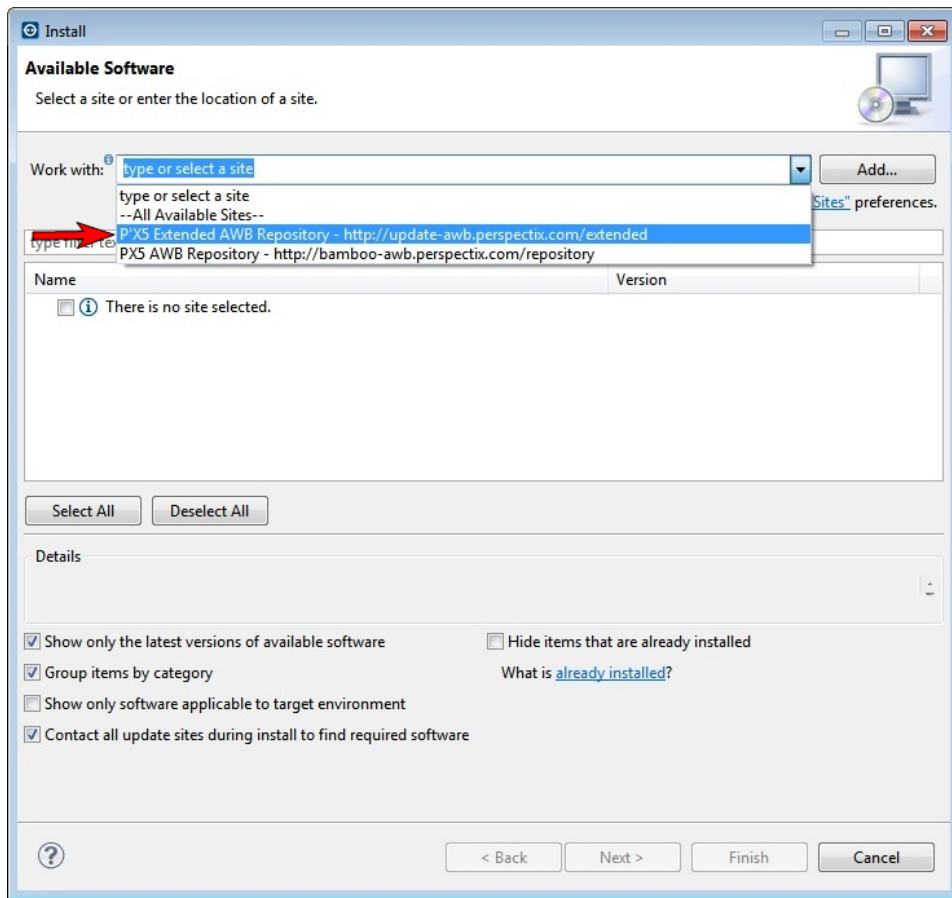


Note: The Cheat Sheets and the context sensitive help are still available, as long as you are online and you have access to <http://docu.px5.ch>.

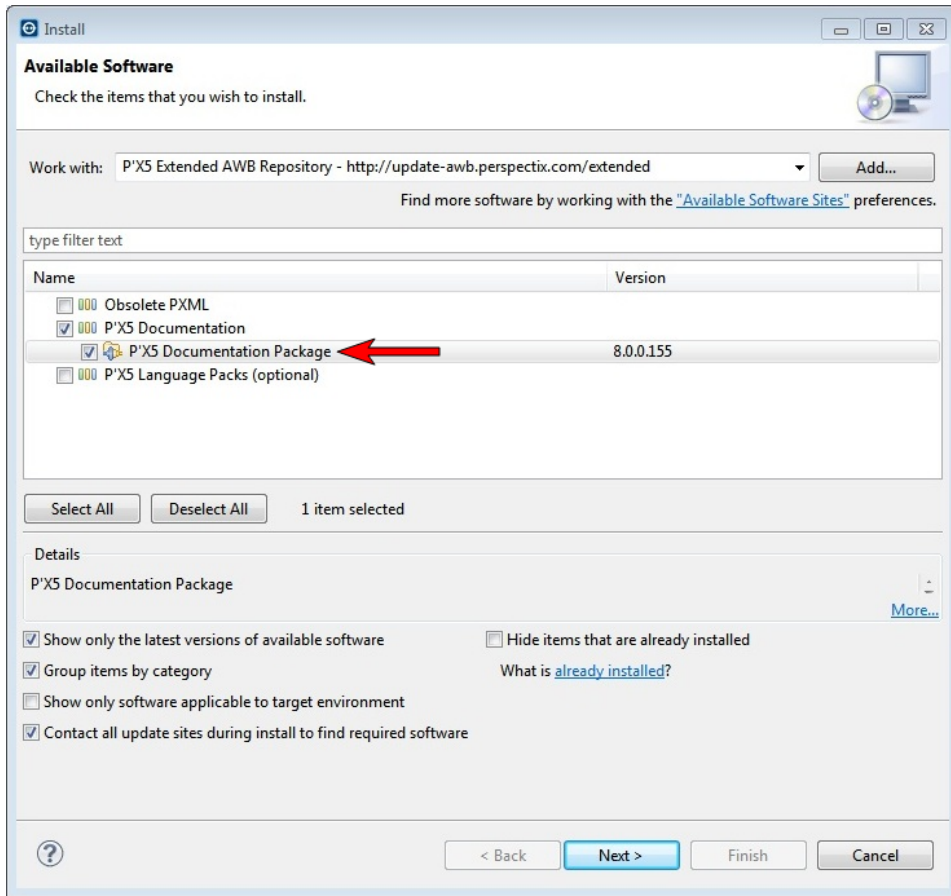
To install the documentation locally

You must install the documentation plugin manually to have local access to the documentation.

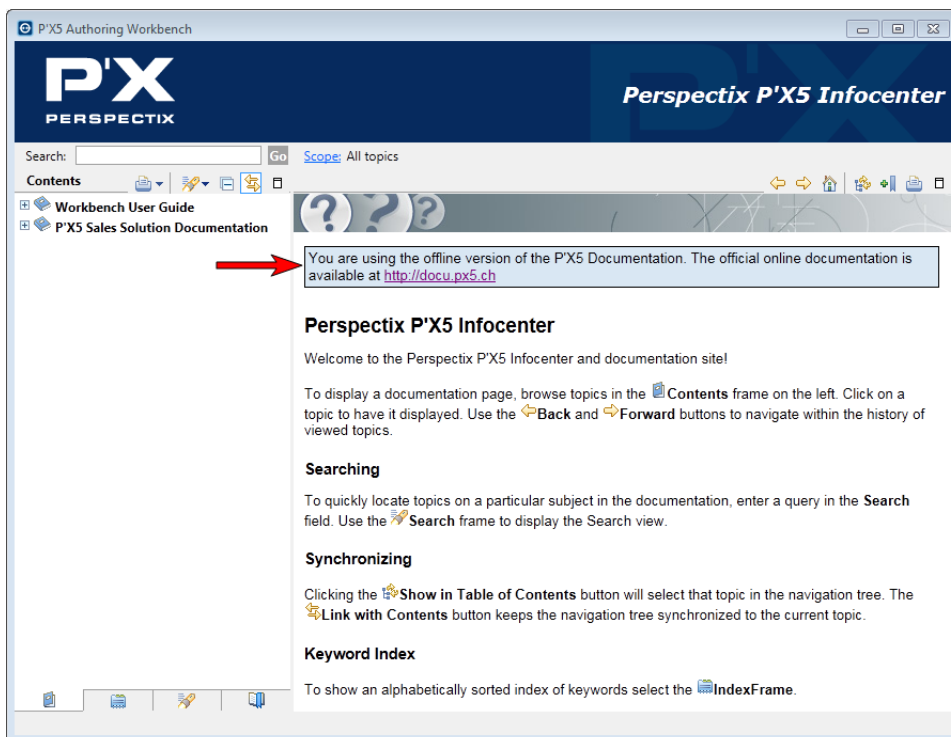
1. Choose **Help > Install New Software**.
2. Select **P'X5 Extended AWB Repository**.



3. Select **P'X5 Documentation Package** and click **Next**.

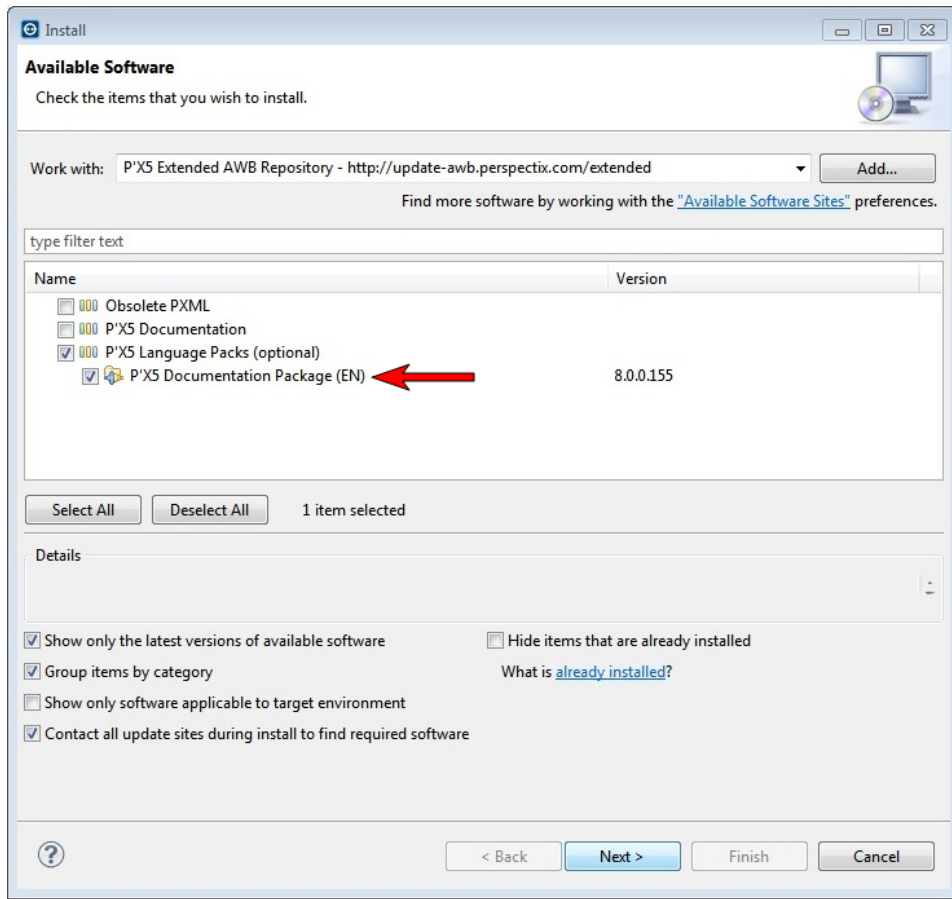


4. Accept the license, wait for the installation to finish and restart the **Authoring Workbench**.
A new message appears on the welcome page when you choose **Window > Help Contents**.



To install a language pack

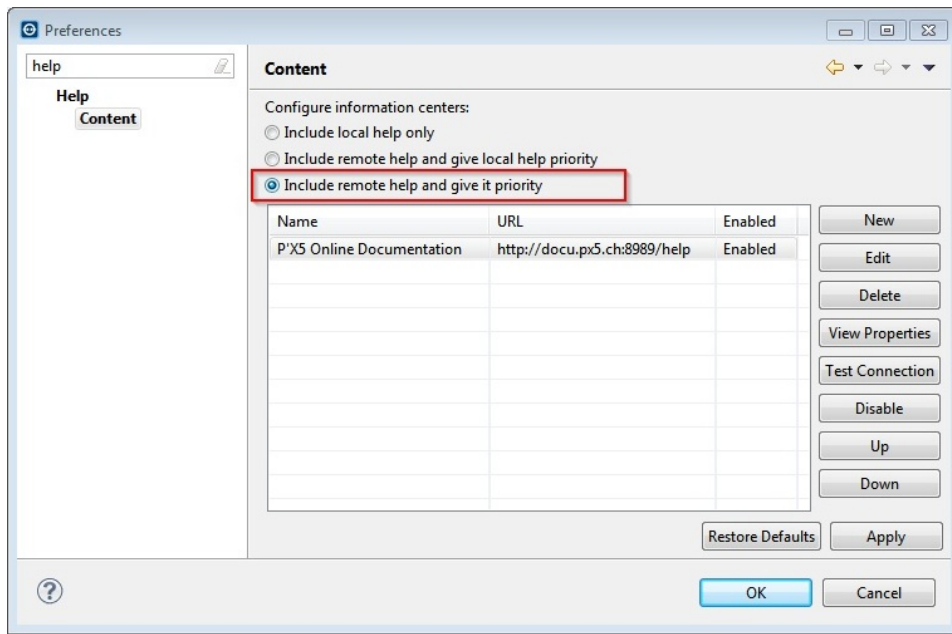
1. Perform the same steps as described in *To install the documentation locally* above.
2. Select the desired language pack, e.g. **P'X5 Documentation Package (EN)**.



To use the online documentation

By default, the locally installed version of the documentation has a higher priority than the online version. You can change this behavior, so that you use the online documentation, and only use the local documentation as a fallback if you have no access to the online documentation.

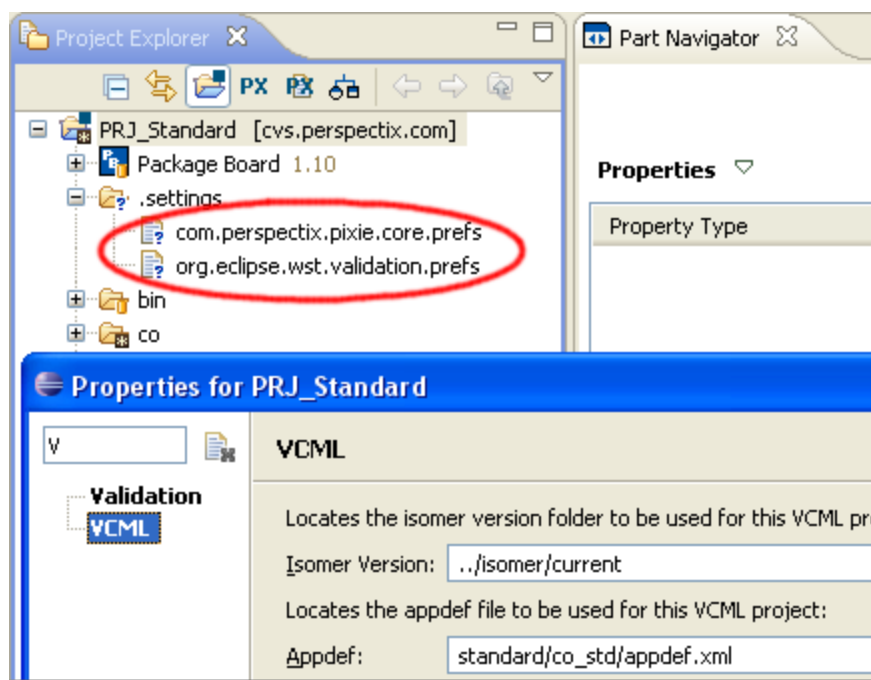
1. Choose **Window > Preferences > Help > Content**.
2. Select **Include remote help and give it priority**.



Share project settings

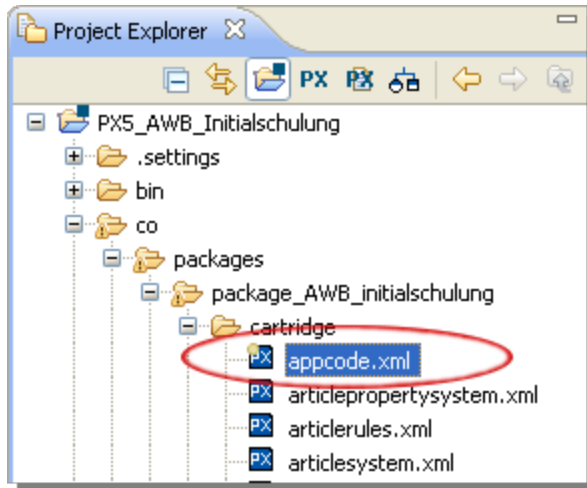
Project settings that are the same for all users of the project are encoded in special files within the `.settings` folder in each project. This folder may be shared using CVS in order to synchronize those project settings with other team members.

Shared settings include the paths set from the project property page and top-down assembly support.



Mark files not read by the VCML builder

You can mark files, which have not been visited by the builder with a small yellow dot.



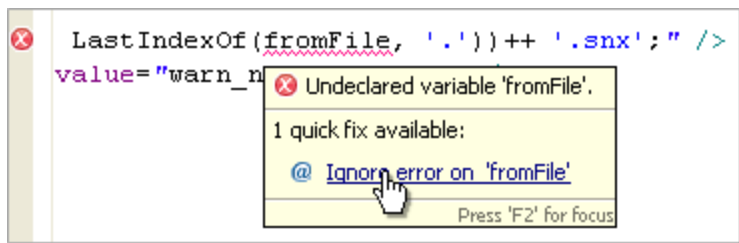
Choose **Window > Preferences** to open the **Preferences** window, then choose **General > Appearance > Label Decorations** and select the **VCML Built Files** checkbox.

Suppress errors

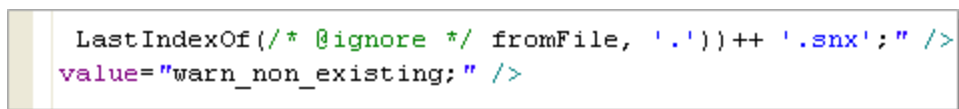
The expression compiler cannot determine the availability of global variables in all occasions, resulting in false positive problem markers for unresolved variables.

If a variable is known to exist, you can mark variables as known and force the compiler to ignore those variables in the expression's context.

Press CTRL+1 (quick fix) or hover the mouse over a problem marker and choose **Ignore error** to ignore such a problem.



The ignored variable:



Note: You can also initiate the quick fix from the **Problems View**. Press CTRL+1 or choose **Quick fix** from the context menu.

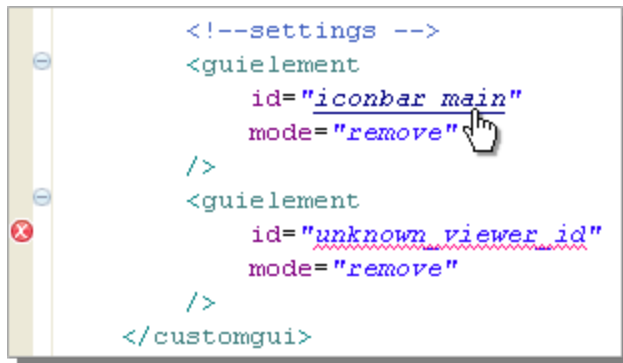
Version: Requires P'X5 version 6.0 or higher.

Resolve viewer IDs

Viewer IDs referenced in `guiElement` elements are parsed and validated by the PXML build.

If a viewer ID cannot be resolved, problem markers appear.

Press and hold down the CTRL key while you click on a viewer ID reference to open the declaration of the referenced viewer.



Note: You must enable **Parse Viewers** in the project's VCML settings for this feature to work.

Version: Requires P'X5 version 6.0 or higher.

Semicolons in expressions

You don't have to terminate blocks in expressions with a semicolon. PXML expressions are similar to other languages like Java or Groovy.

```
def sum(n1, n2) {  
    if (ActualDate() == Date(2010,04,01)) {  
        42;  
    } else {  
        n1 + n2;  
    }  
}
```

You can however still place a semicolon after a block. Also, a semicolon is still necessary to complete an assignment statement, even when the right hand side is a block:

```
def sum(n1, n2) {  
    my result= if (isFoolsDay()) { 42; } else { n1 + n2; };  
};
```

Caution: This feature requires use of the standard library 6.0. Using the Authoring Workbench with an older standard version may result in expressions being accepted by the workbench, but rejected at runtime.

Version: Requires P'X5 version 6.0 or higher.

Simpletext mode in Property Representation

The Propertyrepresentation Editor supports the `simpletext` mode, which displays as a read-only text box. It takes a single argument, `textformula`, which is an expression defining the contents of the text box.

View violations against overwrites and final keywords

The Authoring Workbench reports violations against overwrites and final keywords of expression

functions and operators.

Overwriting a final function is reported as an error.

```
<function>
  <definition
    expression="def useNodeSynchronization() {}"/>
  </function>
```

Function useNodeSynchronization overwrites a final function.

Marking a function as overwrite causes an error if there is no overwrite function.

```
<function>
  <definition
    expression="def overwrites nonExisting() {}"/>
  </function>
```

Function nonExisting is marked as overwrites, but does not overwrite.

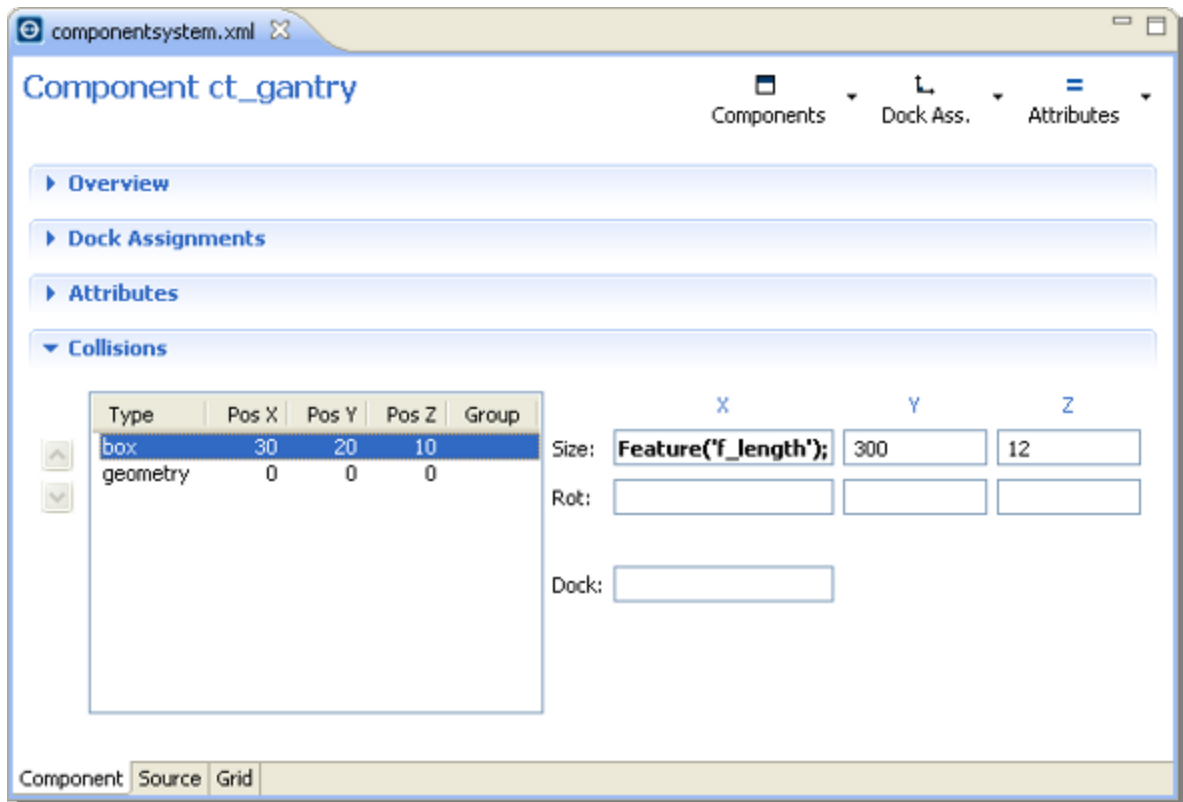
Calling a deprecated user-defined function results in a warning.

```
<function deprecated="true">
  <definition
    expression="def oldFunction() {}"/>
  </function>
<function>
  <definition
    expression="def caller() {
      call oldFunction();
    }"/>
  </function>
```

Deprecated user defined function 'oldFunction' with 0 parameters.

Edit collision assignments

You can assign the collision behavior of components in the ComponentSystem Editor or in the collision assignments file, defined in the package declaration.

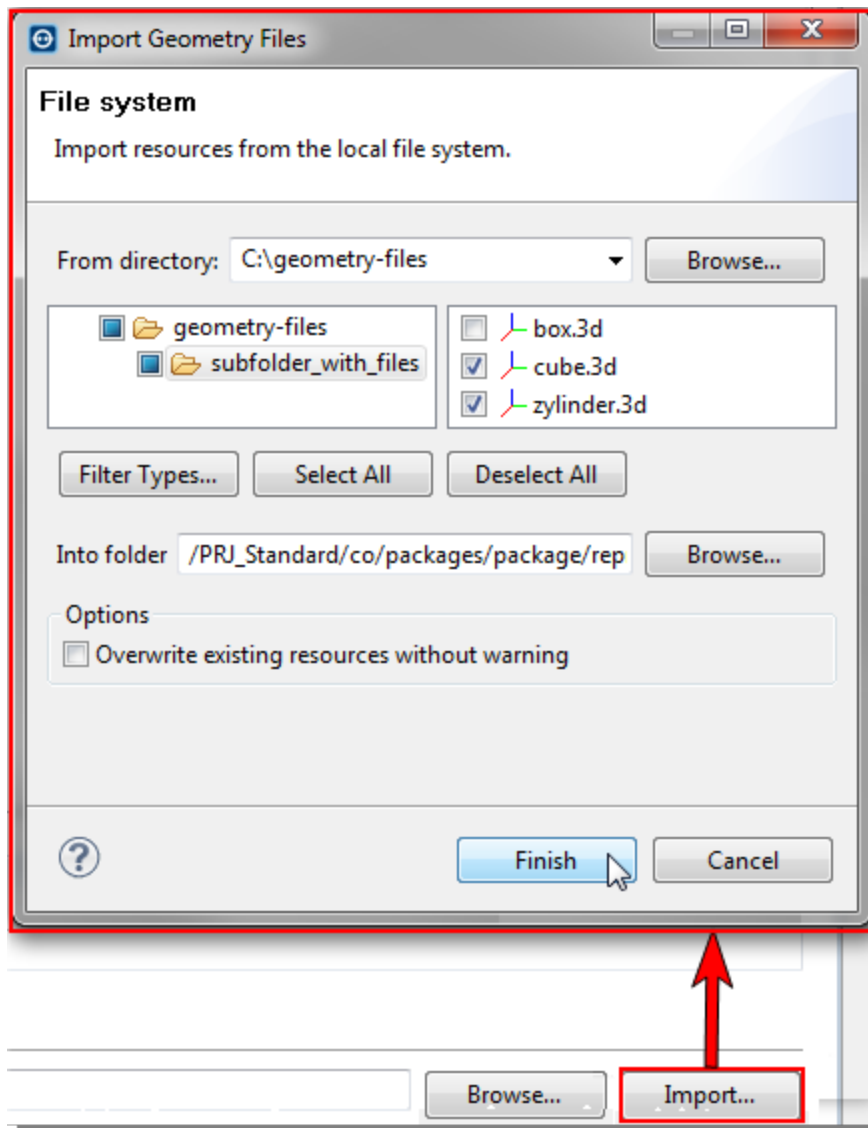


Note: You can copy collision box specifications from the Inspector and paste them into the **Collisions** section.

Import geometry files

To import geometry files in the Geometryrepresentation Editor

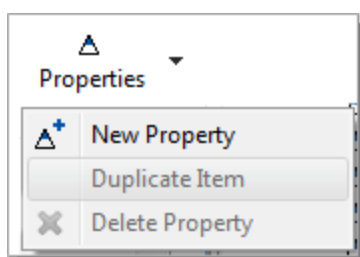
1. In the **Geometryrepresentation** Editor, click the **Geometries** button and then the **Import** button.
2. In the **Import Geometry Files** window, select the geometries you want to import and click **Finish**. You can import entire folder structures and multiple files at once.



Duplicate property element in PropertySystem Editor

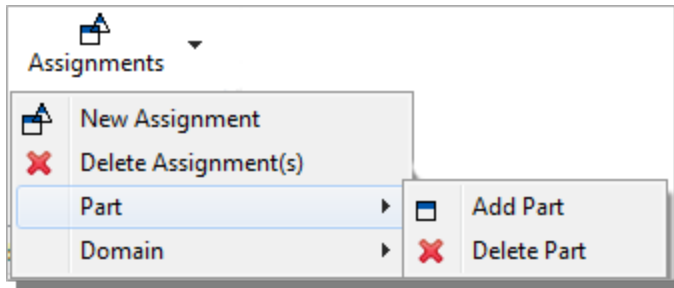
To duplicate a property element in the PropertySystem Editor

- Click the **Properties** button in the **PropertySystem Editor** and choose **Duplicate Item** from the context menu.



To add or delete assignments, parts, and domains on the Property Assignments page

- Click the **Assignments** button on the **Property Assignments** page in the **PropertySystem Editor**.



Version: Requires P'X5 version 6.2 or higher.

Populate discrete domain values

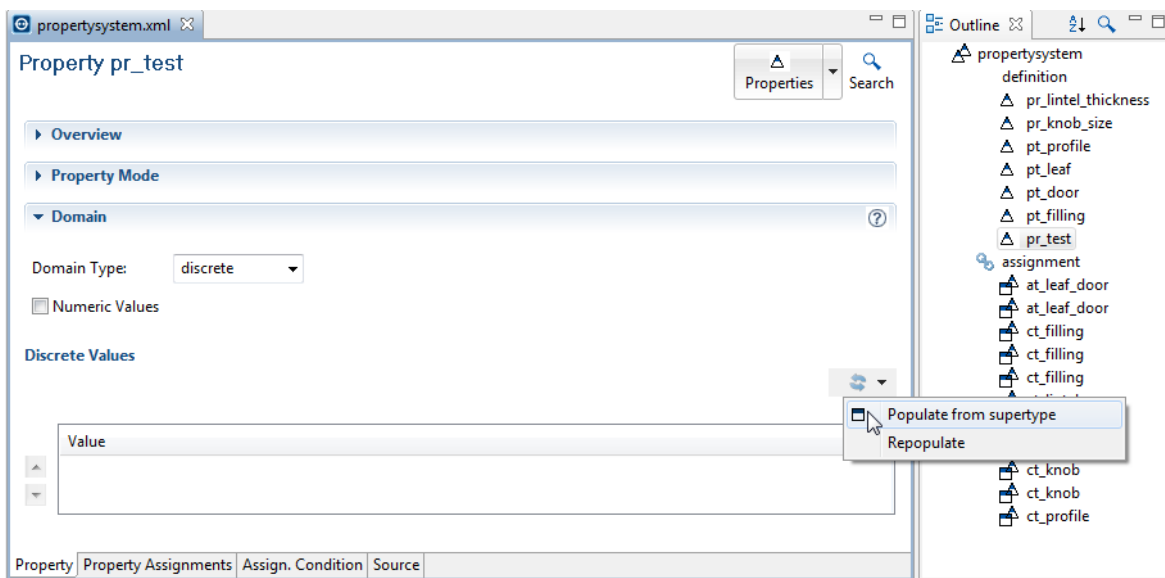
You can easily create discrete domain values when working with morphologies in top-down assemblies. You can populate all values of the domain instead of creating complex active expressions or filling in all component names by hand. Just specify a supertype component and all the names of its children will be automatically added to the value table.

To populate all values of the domain

- Choose **Populate from supertype** from the context menu.

To synchronize the domain definition with the previously selected component supertype

- Choose **Repopulate** from the context menu.



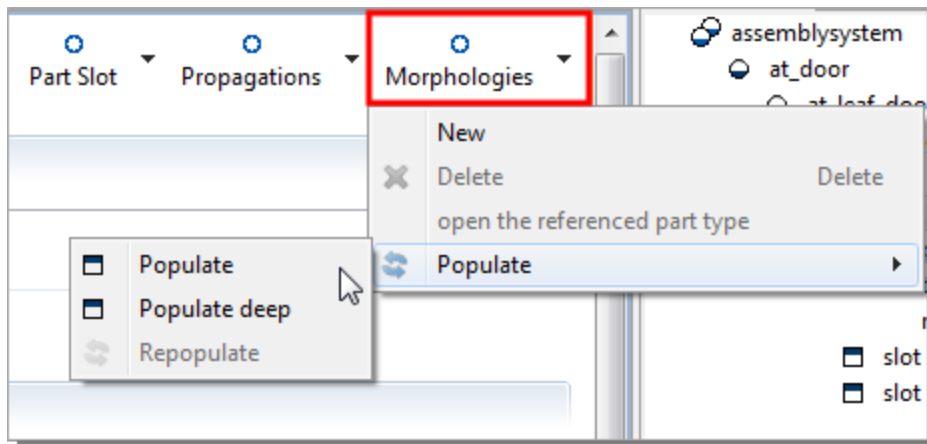
Populate morphology

You can automatically add all leaf elements of a component to the morphology table in the Assembly System. You can either populate the direct children or add all leaf elements recursively. After populating the morphology elements once, simply use the repopulate action to fill-up the table.

To automatically add all leaf elements of a component to the morphology table

Do one of the following:

- Click the **Morphologies** button in the **Assemblysystem Editor** and choose **Populate**.
- Right-click in the **Morphology** table and choose **Populate** from the context menu.



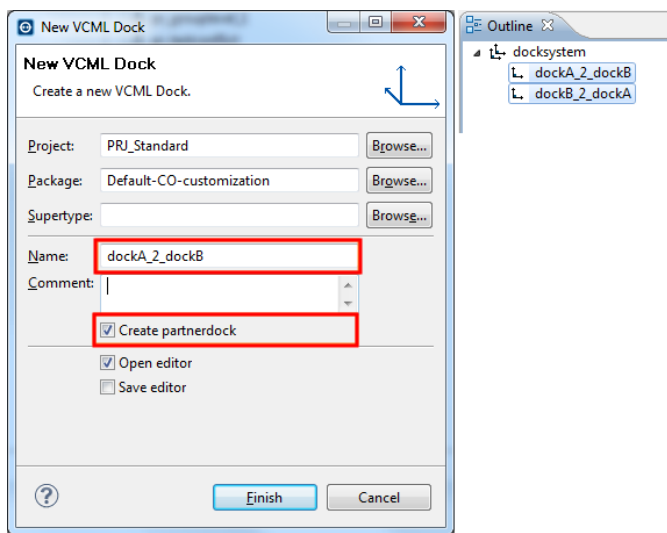
Create partnerdock automatically

To automatically create a partnerdock when you create a dock

- Select the **Create partnerdock** checkbox in the **New VCML Dock Wizard**.

The checkbox is only active if the name of the dock indicates that the dock is a partnerdock:

Pattern	Example
(.*)_2_(.*)	dockA_2_dockB
(.*)2(.*)	dockA2dockB
(.*)_to_(.*)	dockA_to_dockB



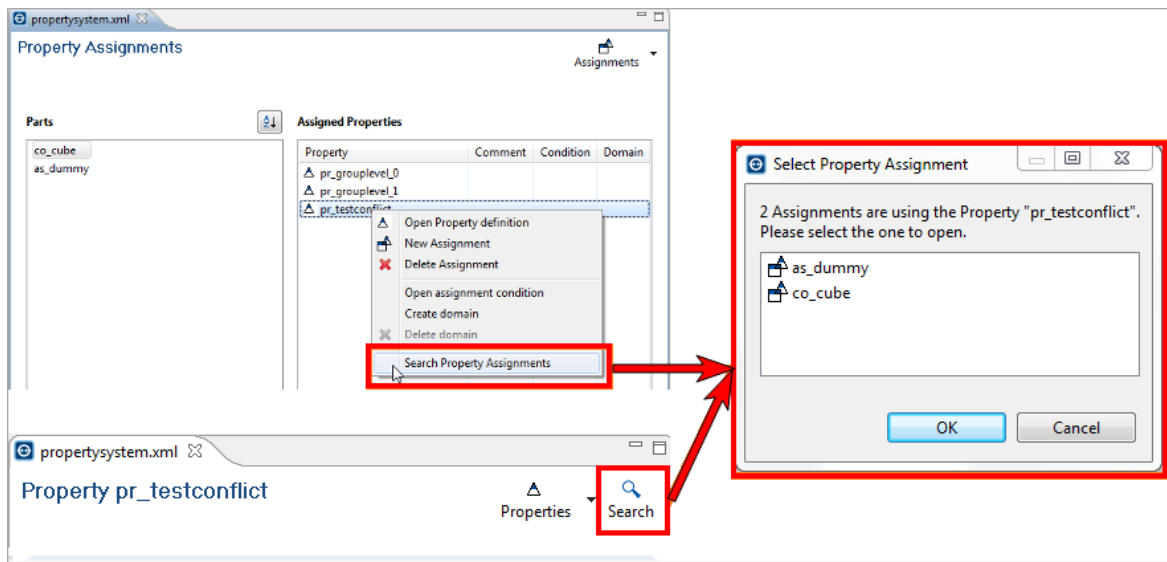
Find property assignments

To find all assignments using a specific property

Do one of the following:

- Right-click an already assigned property in the **Assigned Properties** table and choose **Search Property Assignments** from the context menu.

- Click the **Search** button in the top right corner of the **Propertysystem Editor**.

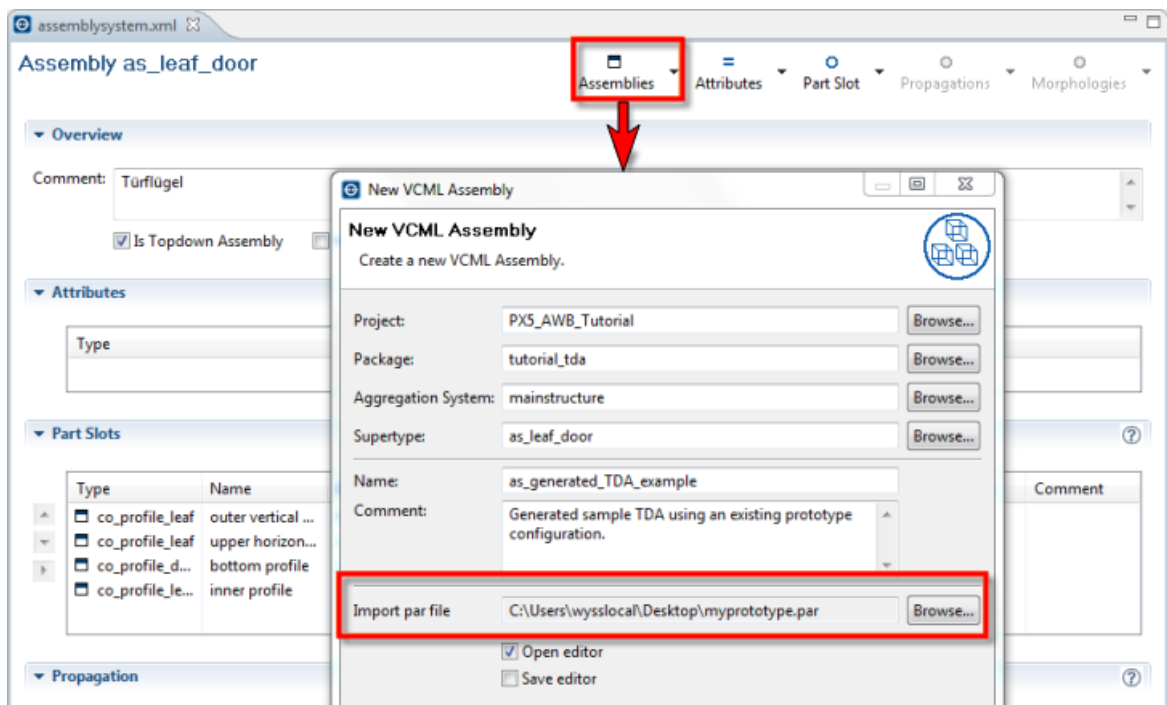


Generate top-down assembly from configuration

You can import a configuration (.par file), which you previously exported from the Configurator, directly into the Authoring Workbench using the New Assembly wizard. This generates all containing attributes automatically and sets the type of your assembly to a top-down assembly.

To automatically generate a top-down assembly from a configuration

- Click the **Assemblies** button in the **Assemblysystem Editor** and select an existing .par file in the **Import par file** field in the **New VCML Assembly** window.



Lazy translation keys

Use the @translate annotation to mark translation keys in expressions without having to localize

them. This can be useful, e.g. if you don't want to localize the key directly but rather use it internally as a parameter value. Using this annotation, such keys can still be found and managed using the Authoring Workbench translation system.

The following example shows the usage of the `@translate` annotation in an expression:

```
my keyInVar = /* @translate */ 'loginviewer.title';
```

The key `'loginviewer.title'` is now accessible via the Authoring Workbench Translation Editor and Externalize Strings tools.